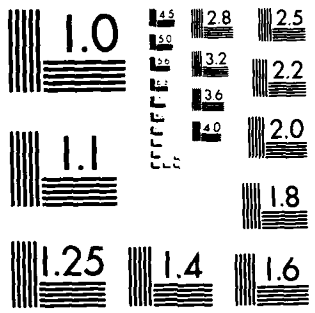
AD-A141 834 A TUTORIAL FOR THE RAMTEK 9460 RASTER GRAPHICS SYSTEM AND THE DI-3000 GRAPHICS PACKAGE(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA R H ELMLINGER MAR 84 1/1

UNCLASSIFIED F/G 9/2 NL

END
DATE FILMED
7-84
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

A TUTORIAL FOR THE RAMTEK 9460 RASTER GRAPHICS SYSTEM AND THE DI-3000 GRAPHICS PACKAGE

by

Ronald H. Elmlinger

March 1984

Thesis Advisor: G. R. Porter

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A141834	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Tutorial for the RAMTEK 9460 Raster Graphics System and the DI-3000 Graphics Package		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; March 1984
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Ronald H. Elmlinger		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		12. REPORT DATE March 1984
		13. NUMBER OF PAGES 88
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

DI-3000, Graphics, High Resolution Graphics, Graphics Tutorial, Ramtek

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document is a tutorial for programming with the DI-3000 graphics software package, Ramtek 9460 graphics hardware, and VAX 11/780 computer located in the Naval Postgraduate School's Wargaming Analysis and Research Laboratory.

For first time users, an introductory level explanation of the functions and terminology associated with the graphics package is presented. This document can also serve as a departure point for programmers who wish to make more extensive use of available capabilities.

A Tutorial for the RAMTEK 9460 Raster Graphics System and the DI-3000 Graphics Package

by

Ronald H. Elmlinger
Lieutenant, United States Navy
B.S., Colorado University, 1977

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY
(Command, Control and Communications)

from the

NAVAL POSTGRADUATE SCHOOL
March 1984

Author: Ronald H. Elmlinger

Approved by: [illegible]
Thesis Advisor

[illegible]
Second Reader

[illegible]
Chairman, Command, Control and Communications Academic Group

[illegible]
Academic Dean

ABSTRACT

This document is a tutorial for programming with the DI-3000 graphics software package, Ramtek 9460 graphics hardware, and VAX 11/780 computer located in the Naval Postgraduate School's Wargaming Analysis and Research Laboratory.

For first time users, an introductory level explanation of the functions and terminology associated with the graphics package is presented. This document can also serve as a departure point for programmers who wish to make more extensive use of available capabilities.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. RAMTEK AND DI-3000 RELATIONSHIP

The Naval Postgraduate School's Secure Wargaming Analysis and Research Laboratory (War Lab), located in Ingersol Hall room IN-157, is equipped with a high resolution graphics system. This system consists of six Ramtek GM859C color monitor screens, four 9460 controllers, and a Precision Visuals Incorporated DI-3000 graphics software system as modified by Lawrence Livermore Laboratories. The DI-3000 runs on a DEC VAX 11-780 using the VMS operating system.

The purpose for writing this tutorial is to provide first-time users in the War Lab with an introductory level guide to using the Ramtek/DI-3000 graphics system, and to serve as a departure point for programmers wishing to make use of the extensive capabilities documented in Reference 1. The facility is available for student and faculty use.

DI-3000 is a device-independent software package. This independence theoretically enables a user to display graphics output on any type of standard graphics device if it is connected to a DI-3000 supported computer system. Because the DI-3000 system does not support all Ramtek features, and because some DI-3000 routines do not work as indicated in Reference 1, another purpose of this guide is to document these differences.

The DI-3000 graphics software package is convenient and easy to use. The only background a programmer needs is an elementary knowledge of Fortran and the ability to create and run a program on the VAX 11-780. This is because DI-3000 is merely a library of Fortran-callable subroutines.

An application program calls DI-3000 subroutines to generate graphics images on one or more of the Ramtek monitors.

The remainder of this chapter provides definitions and conventions, and explains general system use in broad terms. More detailed documentation is given in subsequent chapters.

B. LABORATORY EQUIPMENT OVERVIEW

The secure laboratory as typically configured is shown in Figure 1.1. Variations to this layout occur periodically, but of interest is the location of the six Ramtek monitors. At present, only three unique images can be displayed between these six monitors because of current hardware constraints. For example, monitor pair A and B may display an identical image, monitor pair C and D may share a second image, and monitor pair E and F a third image. Each user specifies within their program the monitor pair to be used to display their graphics output. Only three programs can display their images on the three monitor pairs at the same time.

By energizing just one of the monitors in a pair, for example, A but not B, a single image will appear on A instead of duplicates on both A and B. A further description of how to specify and identify monitors is found in Chapter V.

The VT-100 or VT-102 terminals are the most convenient for use in conjunction with graphics programming. They are equipped with edit function keys and are located near the Ramtek monitor screens and the input graphic tablets. The easiest way to get started in the laboratory is to obtain an account on the VAX system from the War Lab Manager and review the laboratory familiarization handout written by LCDR McCoy. Copies of this handout are available in the War Lab. Completing the CAI Tutorial is also helpful so that unique features of the VAX editor can be used to create DI-3000 Fortran programs. The CAI tutorial is an interactive program found on the VAX system. Its beginning instructions will appear on the terminal screen immediately after log-in.

C. RUNNING A PROGRAM

After a Fortran program file is created that consists of DI-3000 subroutine calls which display a graphics image, the file must be compiled, linked to DI-3000, then run in three distinct steps. Assume you have created a program file identified as PICTURE.FOR;5, and you wish to run the program and display the image on a Ramtek monitor. First you would type

```
FOR PICTURE
```

This command would compile your latest version Fortran file, here number 5, named PICTURE. If compilation were successful, the command prompt

```
$
```

would appear. If not, errors would result. To link the DI-3000 operating system to your program you would next type

```
SLINK PICTURE
```

If there were no errors in your use of DI-3000, you would again see the command prompt. Then you would run the program by typing

```
RUN PICTURE
```

Figure 1.2 is an example of the control commands and screen returns that will be displayed when successfully running PICTURE.FOR;5 with no errors. Notice that there are five compilation warnings listed. These are normal system responses. If other than these warnings are listed, you have generated an error condition either in your use of DI-3000 or during the compilation and run phase of your Fortran program.

Once a program has been successfully run at least once, an object file is created. Any time the user would then desire to display the image, only "running" the program would be necessary. For our previous example, only the command

```
RUN PICTURE
```

would be required.

D. INITIALIZATION OVERVIEW

The fundamental DI-3000 subroutine calls define primitive objects and images such as lines, polygons, moves to a new screen position, written text, and more. There are three classes of primitives: polylines, polygons, and text. Polyline primitives form "open-ended" figures, while polygon primitives form "enclosed" ones. Text primitives generate written character information. A more detailed description of primitives and their applications can be found in Chapters II, III, and IV.

Every DI-3000 program must also consist of other subroutine calls necessary for these image primitives to be created, displayed, or terminated. For the programmer's purposes, this means that each program must have a certain minimum sequence of commands for it to run successfully. Figure 1.3 is an example of a DI-3000 program harness that will display a picture if the image primitive calls are surrounded by the commands listed. These commands are a minimum of the calls a programmer can use, and provide a good reference for the beginning user. A full explanation of each can be found in Chapter V.

E. PRIMITIVES AND COMPOUND IMAGES

To create, say, a polygon primitive we would call a DI-3000 Fortran subroutine, named JPOLGN, as follows:

```
CALL JPOLGN(X,Y,N)
```

Notice the existence of subroutine parameters X,Y,N. These parameters mean the following:

X - The X coordinate positions (real array)

Y - The Y coordinate positions (real array)

N - Number of points defining a polygon (integer)

Also, notice that subroutine parameters are of varying types (real array and integer). In accordance with Fortran convention, this means parameters must be declared and dimensioned correctly at the beginning of your program. Real, integer, and array parameters are discussed further in Paragraph G below. A complete description of the subroutine JPOLGN can be found in Chapter II.

The degree of complexity of an object being created determines the complexity of its primitive call. Objects such as polygons are many-sided, can have different interior and edge colors, and different interior intensities. These characteristics of a primitive are examples of its attributes. Notice that we did not specify the attributes of the polygon when we called JPOLGN above, therefore we would have no way of knowing characteristics like what color polygon we would create or what the interior pattern would be. We must insure that these attributes are correctly defined. There exist, for this purpose, attribute subroutines that are normally invoked prior to primitive subroutines. For example, to insure a solid interior pattern for all subsequent polygons, we would include in our program the following subroutine call:

```
CALL JPINTR(1)
```

This attribute subroutine and others that are commonly used are discussed in detail in Chapter III.

Seldom is any image created that merely consists of a single primitive shape. Rather, most images are composites of several primitives that build upon each other to create the final desired picture. Most of the differences between the way the DI-3000 system is supposed to work and the actual results obtainable in the War Lab are encountered when the programmer tries to create and display compound image segments. A discussion of the differences, their limitations, and ways to achieve desired outputs are described in Chapter VI.

F. ABSOLUTE AND RELATIVE COORDINATES

DI-3000 subroutines create images for display at a defined location on an output device. Most primitive subroutine calls can create images based upon either <u>absolute</u> or <u>relative</u> coordinate positions. When an image is to be created only once, use of either absolute or relative positioning is appropriate depending on the user's application and preference. But when multiple copies of an image are desired, it would be needlessly repetitive to redefine each primitive image for each different position, therefore relative referenced primitives are used. Rather than using absolute coordinates the relative primitive creates an image at X,Y positions relative to an imaginary, unseen marker on the screen. For example, to duplicate an image you would establish the marker, call the relative primitive, move the marker, call the relative primitive again, and repeat this sequence until all images were created.

Primitives of absolute and relative parameters can be distinguished from each other by their formats. Absolute primitives are of the form:

JXXXXX (e.g. JPOLGN)

while relative primitives are of the form:

JRXXXX (e.g. JRPLGN)

Almost every attribute and capability of both are identical, except for the coordinate position differences.

G. CONVENTIONS

1. General

Throughout this tutorial, Reference 1 terminclogy and fcrmat has been maintained so that the programmer can use both documents interchangably when creating Ramtek graphics images. Answers to questions regarding standard VAX Fortran conventicn can be found in Reference 2, located in the War Lab.

2. Display Coordinate Systems

All images are referenced to a world and a virtual coordinate system. For our purposes, we will define the world cocrdinate system to be ±10.0 units in the horizontal (X) direction, and ±8.0 units in the vertical (Y) direction. Therefore, the program grid is a rectangular display area 20.0 units wide by 16.0 units high. Except for maintaining the height-to-width ratio of a Ramtek GM859C monitor (.8 to 1.0), cur choice cf dimensions is completely arbitrary. Whenever a primitive image is defined in our program, it will be referenced tc our 16.0 by 20.0 unit world coordinate grid.

We will also use the entire available Ramtek monitor screen surface. This virtual coordinate system relative to screen center will be a rectangular grid ±1.0 unit wide by ±.8 units high.

Cur DI-3000 program will "translate," or "map" our world coordinate picture onto the virtual coordinate screen by using the JWINDO command to specify world coordinate axis values, and the JVSPAC command for virtual coordinate values. The use of these commands is described in Chapter V.

3. Subroutines and Parameters

Each DI-3000 subroutine will be described in the section appropriate to the image type created. Each call will be briefly described as to function and result, and its parameters will then be listed and defined as to meaning and type. The correct specification for real number, integer, and array variables within a DI-3000 Fortran program follows:

Integer--A whole number (no decimal point)

Real Number--A number with a decimal point.

Array--A group of contiguous storage locations associated with a single symbolic name [Ref. 2]. It must be dimensioned at the beginning of the Fortran program, and it can be integer or real as declared.

Integers and real numbers can also be variable names referencing integer and real numbers, respectively.

4. Current Position

Primitives use and often modify the current position (CP), a world coordinate point that determines a "starting point." Unless otherwise specified, a description of a primitive subroutine call will assume the return of the CP to its position prior to the call.

If the CP is placed outside the defined coordinate area, distorted images may result when a drawing attempt encounters a defining screen edge. In general, unless special effects are desired, it is best to insure all created images will "fit" on the screen.

5. Attributes

Attributes associated with a primitive subroutine call will be listed by attribute class. Recall, primitives are classified as polylines, polygons, or text and their attributes are specified based upon these primitive types. System default settings (settings that are initialized if no attribute is specified) are also found in Chapter III.

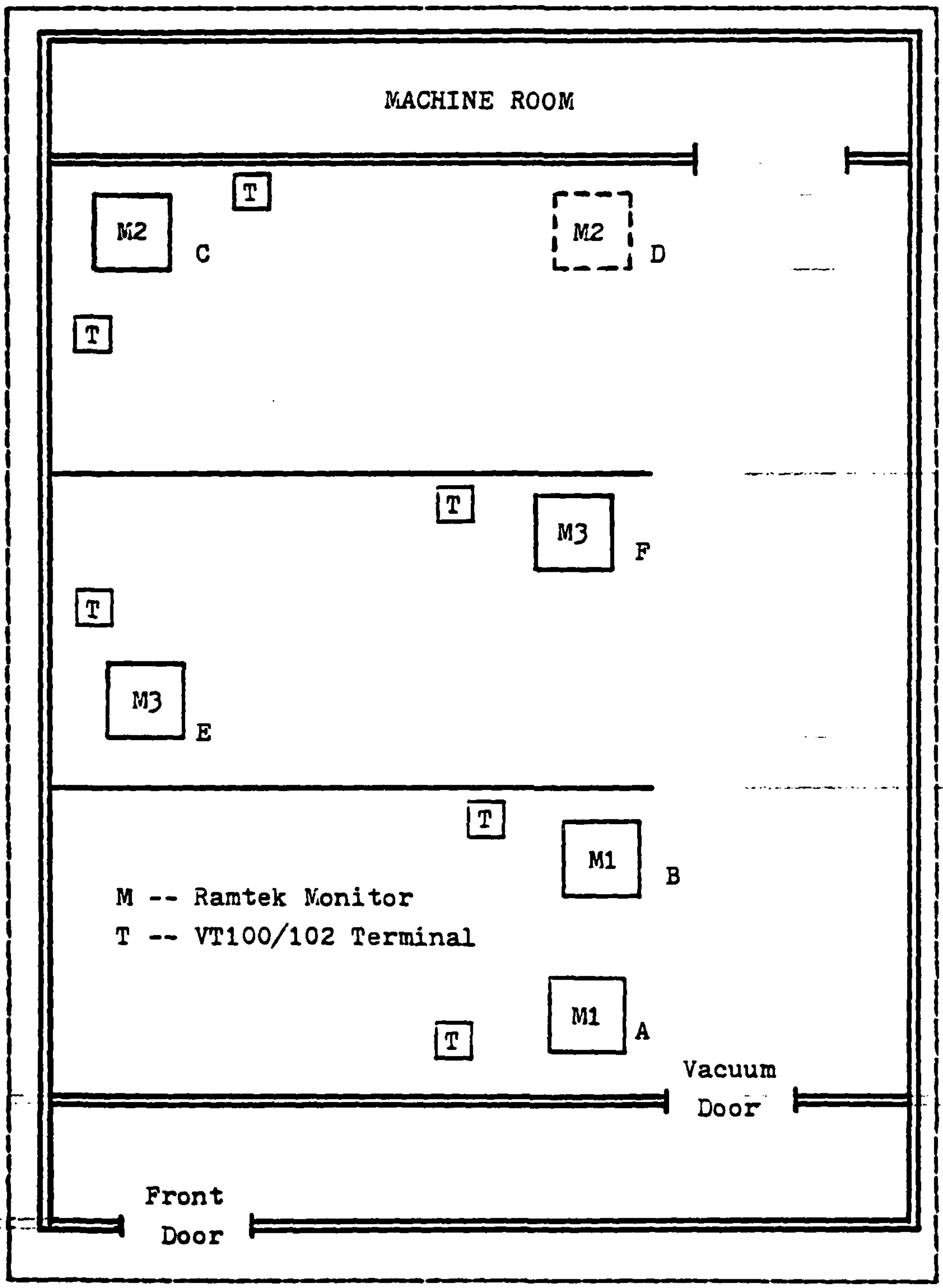


Figure 1.1 Typical War Lab Configuration.

```
$ FOR PICTURE
$ SLINK PICTURE
%LINK-W-WRNERS, compilation warnings
        in module DI3 file SYS$SYSROOT:[SYSLIB]DI3000N.EXE;73
%LINK-W-WRNERS, compilation warnings
        in module DD1NOD file SYS$SYSROOT:[SYSLIB]DD1NOD.EXE;71
%LINK-W-WRNERS, compilation warnings
        in module DD2NOD file SYS$SYSROOT:[SYSLIB]DD2NOD.EX2;67
%LINK-W-WRNERS, compilation warnings
        in module DD3NOD file SYS$SYSROOT:[SYSLIB]DD3NOD.EX2;67
%LINK-W-WRNERS, compilation warnings
        in module DD4NOD file SYS$SYSROOT:[SYSLIB]DD4NOD.EX2;67
$ RUN PICTURE
```

Figure 1.2 VAX Commands and Returns for Program Run.

```
C PROGRAM HARNESS EXAMPLE
C
      INTEGER MON
C
C INITIALIZE A RAMTEK MONITOR
C
      TYPE *, 'ENTER THE MONITOR NUMBER'
      ACCEPT *, MCN
C
C BEGIN MANDATORY INITIALIZATION CALLS
C
      CALL JBEGIN
      CALL JFILES(3,1,MON)
      CALL JDINIT(1)
      CALL JDEVON(1)
      CALL JDCOLR(2)
      CALL JVSPAC(-1.0,1.0,-.8,.8)
      CALL JWINDO(-10.0,10.0,-8.0,8.0)
      CALL JROPEN(1)
C
C          .
C          .
C          .
C (INSERT YOUR PRIMITIVE CALLS HERE FOR
C IMAGE CREATION)
C          .
C          .
C          .
C (ALSO, INSERT YOUR REQUIRED ATTRIBUTES
C BASED UPON ATTRIBUTE CLASS AS DESCRIBED
C IN CHAPTER III.)
C          .
C          .
C ALL PRIMITIVE CALLS INSERTED HERE ARE
C WITHIN OPEN RETAINED SEGMENT NUMBER 1.
C          .
C
      CALL JRCLOS
      CALL JPAUSE(1)
      CALL JEND
      END
C
C THIS PROGRAM DRAWS IN DEFAULT GREEN AND USES A
C RETAINED SEGMENT (NUMBER 1).
C
```

Figure 1.3   Minimum Necessary DI-3000 Subroutine Calls.

II. NON-TEXT PRIMITIVES

A. GENERAL

Primitive routines are used by the programmer to describe objects. Character-based objects (text) and graphics objects (non-text) are the two most common primitive types. They are distinguished not only by the images they create, but by how they are specified and used. This chapter describes non-text primitives. Chapter IV provides details to display text.

As discussed in Chapter I, DI-3000 non-text primitives often define images in one of two ways. A figure can be defined using absolute world coordinate position values, or it can be expressed in world coordinate position values relative to the current position (CP). First some elementary primitive subroutine calls will be described followed by examples that will illustrate the difference between absolute and relative expressions.

All primitives will be created based upon the most recent attribute definitions. For example, if we specify line color to be blue, then all lines, figures, and text will be drawn in blue. If we wish to draw a blue line and then a red line, we must set the line color attribute to blue, draw the blue line, change the line color attribute to red, then draw the red line.

Non-text primitives and their attributes are grouped into two major classes: polyline and polygon. Their differences are described in Chapter III.

B. MOVING THE CURRENT POSITION (JMOVE/JRMOVE)

1. Description

JMOVE/JRMOVE are the subroutines used to move an invisible reference point from the current position to a new current position. The commands establish a reference position.

2. Use

```
CALL JMOVE(X,Y)    (Absolute)
      or
CALL JRMOVE(DX,DY)    (Relative)
```

3. Parameter Definition

X,Y (Real)

The world coordinate position that will become the new current position.

DX,DY (Real)

The amount of displacement from the previous position to the new current position.

4. Discussion Example

If the current position is at world coordinate position (1.0,2.0), and we want to establish a new CP at (3.0,7.0) we would use <u>either</u> of the following subroutine calls:

```
CALL JMOVE(3.0,7.0)
      or
CALL JRMOVE(2.0,5.0)
```

Note that the relative call JRMOVE merely <u>adds</u> its parameter values to the current position. For example, 1.0+2.0=3.0 and 2.0+5.0=7.0 results in the new X,Y world coordinates of (3.0,7.0). The absolute call JMOVE changes position directly to its parameter values.

Current position can be moved to X,Y coordinates outside the specified viewport (for example, X>10.0 or Y>8.0). This technique is often useful for creating unusual images such as "gradual" arcs of large radius, but distortion will result when the created shape reaches the viewport boundary.

C. DRAWING A LINE (JDRAW/JRDRAW)

1. Description

JDRAW/JRDRAW are the subroutines used to draw a visible line from the current position to a new current position.

2. Use

```
CALL JDRAW(X,Y)    (Absolute)
        or
CALL JRDRAW(DX,DY)    (Relative)
```

3. Parameter Definition

X,Y (Real)

The world coordinate position that will become the new current position. A line is drawn from the old current position <u>to this new</u> CP.

DX,DY (Real)

The amount of displacement from the previous current position to the new current position. A line is drawn from the old CP <u>to this new</u> CP.

4. Required Attributes

Polyline class.

5. Discussion Example

If the current position is at world coordinate position (-1.0,-4.0) and we want to draw a line from there to new position (2.0,0.0), then to a third and final position (5.0,-2.0) we could use either of the following subroutine call sequences:

```
CALL JDRAW(2.0,0.0)
CALL JDRAW(5.0,-2.0)
        or
CALL JRDRAW(3.0,4.0)
CALL JRDRAW(3.0,-2.0)
```

Any correct combination of absolute and relative calls could also have been used. For example:

```
CALL JDRAW(2.0,0.0)
CALL JRDRAW(3.0,-2.0)
```

Figure 2.1 shows the picture that would result from any of the above three call sequences.

The attributes of the line would correspond to those attributes already established prior to invoking the JDRAW/JRDRAW primitives.

Note that a color attribute must be specified by using the JCOLOR/JDCOLR routines as described in Chapter III.

CP is changed to the final X,Y coordinate position of the drawn line.

D. DRAWING CONNECTED LINES (JPOLY/JRPOLY)

1. Description

JPOLY/JRPOLY are the subroutines used to draw a connected sequence of visible lines, known as a poly<u>line</u> segment.

Note: the subroutines do <u>not</u> create poly<u>gons</u>.

2. Use

CALL JPOLY(X,Y,N) (Absolute)

or

CALL JRPOLY(DX,DY,N) (Relative)

3. Parameter Definition

X,Y (Real, Array)

The arrays contain the absolute world coordinate positions that define the points of the polyline.

DX,DY (Real, Array)

The arrays contain the relative amount of displacement from the previous polyline point to the next point that defines the polyline.

N (Integer)

The number of points in the polyline segment (not to include the first current position point).

The above arrays must be dimensioned to at least the value of N.

4. Required Attributes

Polyline class.

5. Discussion Example

Polylines and polygons differ in that polygons are always enclosed figures. Polygons are also "filled" in color and will be described later in this chapter.

Polylines are created using a single JPOLY/JRPCLY subroutine call rather than a sequence of JDRAW/JRDRAW calls as before.

Once again we will create the image shown in Figure 2.1, but this time only a single subroutine will be needed. Notice, though, that now the defining polyline coordinate pairs are held in array variables. Also note that the initial polyline position is defined as the current position, (-1.0,-4.0) again, and is not included in the array variable coordinate pairs. Either of the following subroutines will define the required connected line:

```
CALL JPOLY(X,Y,2)
```

where X and Y are both two element arrays with the following values:

```
X(1)=2.0    Y(1)=0.0
X(2)=5.0    Y(2)=-2.0
```

or

```
CALL JRPOLY(DX,DY,2)
```

where DX and DY are both two-element arrays with the following values:

```
DX(1)=3.0    DY(1)=4.0
DX(2)=3.0    DY(2)=-2.0
```

The above arrays must be dimensioned in the calling program in accordance with standard Fortran. For example:

```
DIMENSION X(2),Y(2),DX(2),DY(2)
```

or

```
REAL X(2),Y(2),DX(2),DY(2)
```

The N variable is defined as the dimension of the X,Y or DX,DY arrays. Every sequence of lines consists of a beginning point, a series of "breaking" points, and an end point. The beginning point is the current position and is not included in the array variables. All other coordinate pairs are included.

CP is changed to the last X,Y coordinate polyline point.

E. DRAWING AN ARC LINE (JARC)

1. Description

JARC is the subroutine used to draw an arc of a circle. An arc line is drawn counterclockwise from one angle position to another, with a specified radius, from a specified invisible circle center.

2. Use

```
CALL JARC(X,Y,0.0,RADIUS,NSEG,A0,A1)
```

3. Parameter Definition

X,Y (Real)

The world coordinate center of the circle from which the arc is to be drawn.

0.0 (The real number zero)

RADIUS (Real)

The radius of the arc in world coordinates.

NSEG (Integer)

The number of line segments to be used in drawing the arc. If NSEG<1 the arc will be smooth because the maximum number of segments possible will be used to draw a smooth curve. If NSEG>=1 then the arc will be drawn with NSEG flat, straight lines.

A0,A1 (Real)

The angles in degrees defining the span of the arc. Positive angles are measured counterclockwise from the positive X axis of the world coordinate system. Arc lines are drawn counterclockwise from angle A0 to A1.

4. Required Attributes

Polyline class.

5. Discussion Example

The JARC and JSECTR subroutines are very similar, and care must be taken to avoid confusing them. JARC does not draw a filled wedge, but merely defines a section of circle edge (the outside arc). If a "pie shaped" wedge is desired, then the subroutine JSECTR is used.

If a 360 degree circle is desired, then parameters A0 and A1 can be specified close enough together so that any space between them will not appear on the screen. Their values will be world coordinate dependent. An easier way to create a circle is to use the JCIRCL subroutine and specify "no polygon fill" as described later in this chapter.

Note that an arc should lie within the already defined world coordinate window if it is to be drawn undistorted. This means that very large, gradual arcs cannot be created simply by defining a center far away from the image ("off the screen"). Though the large arc will have the correct shape through its curvature, if it exceeds the world coordinate viewing area it will distort at the boundary edge. The technique of viewing transformation should be used to avoid this distortion, an explanation of which can be found in Chapter V.

F. DRAWING A POLYGON (JPOLGN/JRPLGN)

1. Description

JPOLGN/JRPLGN are the subroutines used to draw filled or unfilled polygons. All defined points and the CP are connected to form an enclosed figure. This is accomplished by an "implicit" final draw from the last specified point in the polygon to the initial polygon creation point (CP). The image can be filled with a color and pattern as described in Chapter III.

Note that the subroutines can create polylines if the "no fill" attribute is specified, but unlike polylines, polygons will always define an enclosed figure.

2. Use

```
CALL JPOLGN(X,Y,N)    (Absolute)
          or
CALL JRPLGN(DX,DY,N)  (Relative)
```

3. Parameter Definition

X,Y (Real, Array)

The arrays of world coordinate absolute positions that define the points of the polygon.

DX,DY (Real, Array)

The arrays of displacement from the previous position to the points that define the polygon.

N (Integer)

The number of points in the polygon. The above arrays must be dimensioned by at least the value of N.

4. Required Attributes

Polygon class.

5. Discussion Example

A polygon is defined as a move from the current position to the first point (as specified by the array variables X(1),Y(1) or DX(1),DY(1)), draws to the remaining array points, and a final implicit draw from the last point back to the first point.

Assume we wish to draw the five sided polygon shown in Figure 2.2. Let current position be given initially as the absolute world coordinate position (1.0,2.0). Either of the following subroutines will create the polygon.

```
CALL JPOLGN(X,Y,5)
```

where X and Y are both five element arrays with the following values:

X(1)=2.0	Y(1)=3.0
X(2)=3.0	Y(2)=4.0
X(3)=5.0	Y(3)=5.0
X(4)=6.0	Y(4)=3.0
X(5)=4.0	Y(5)=1.0

or

```
CALL JRPLGN(DX,DY,5)
```

where DX and DY are both five element arrays with the following values:

DX(1)=1.0	DY(1)=1.0
DX(2)=1.0	DY(2)=1.0
DX(3)=2.0	DY(3)=1.0
DX(4)=1.0	DY(4)=-2.0
DX(5)=-2.0	DY(5)=-2.0

Notice that relative array values merely add to the previous position value to create a new coordinate position.

When JPOLGN is finished, the current position (CP) is set to the value of X(1),Y(1), the first point, but JRPLGN returns the initial CP.

Polygons can be convex or concave. Their defining points can create intersecting lines, but the created shape will not correctly "fill" in most of those cases. Polygons must have at least three points.

Note that absolute poly<u>gon</u> routines do not use the CP at all. They require X,Y array specification of all points. Poly<u>line</u> routines use the CP as their first point, and do not include it in their X,Y arrays.

G. DRAWING A RECTANGLE (JRECT)

1. <u>Description</u>

JRECT is the subroutine used to draw a horizontal/ vertical rectangular polygon. It is an easier subroutine to use than JPOLGN because only two X,Y position pair variables are required; the diagonally opposite corners of the rectangle.

2. <u>Use</u>

```
CALL JRECT(X0,Y0,X1,Y1)
```

3. <u>Parameter Definition</u>

X0,Y0 (Real)

One corner of the rectangle.

X1,Y1 (Real)

The diagonally opposite corner of the rectangle.

4. <u>Required Attributes</u>

Polygon class.

5. <u>Discussion</u>

JRECT is included in this text as an effort-saving subroutine for creating horizontal/vertical rectangles. Note that the variables are not arrays, but merely values. Also, there is no counterpart subroutine with opposite corners defined relative to the CP. If more than one identical rectangle is required, the JRPLGN subroutine must be used.

H. DRAWING A CIRCLE (JCIRCL)

1. <u>Description</u>

JCIRCL is the subroutine used to draw a circular polygon. The created shape is connected to form an enclosed figure. The image can be filled with color and pattern. Outlined circles can be created if no fill is specified.

2. <u>Use</u>

```
CALL JCIRCL(X0,Y0,0.0,RADIUS,NSEG)
```

3. <u>Parameter Definition</u>

X0,Y0 (Real)

The center position of the circle in world coordinates.

0.0 (The real number zero)

RADIUS (Real)

The radius of the circle.

NSEG (Integer)

The number of line segments to be used when drawing the circle. If NSEG<3 a smooth edged circle will be created. If NSEG>=3, the outer edge of the circle will be drawn with NSEG flat, straight lines.

4. Required Attributes

Polygon class.

5. Discussion Example

Figure 2.3 shows examples of figures that can be created using the JCIRCL subroutine. The program segment shown in Figure 2.4 was used to create Figure 2.3. Notice that triangles, squares, and any N-sided figure can be formed by altering the value of parameter NSEG. All images are oriented towards the X axis. For example, the first triangle tip lies on the axis at point (RADIUS,0.).

Also note that a smooth circle was formed by setting NSEG=1, and that all the examples were drawn using the same center point and with the "no fill" attribute specified.

For creation of circle portions, like "pie slices" of a circular polygon, refer to the JSECTR subroutine. For simple partial arc lines refer to the JARC subroutine.

I. DRAWING A CIRCLE SECTION (JSECTR)

1. Description

JSECTR is the subroutine used to draw a section of a circular polygon. This shape is often used in the creation of "pie charts", where part of a filled circle is needed.

2. <u>Use</u>

```
CALL JSECTR(X0,Y0,0.0,RADIUS,NSEG,A0,A1)
```

3. <u>Parameter Definition</u>

X0,Y0 (Real)

The world coordinate center of the circle from which the section is to be drawn.

0.0 (The real number zero)

RADIUS (Real)

The radius of the circle section.

NSEG (Integer)

The number of line segments to be used when drawing the outer arc portion of the section. If NSEG<1 a smooth edge will be created. If NSEG>=1, the outer arc will be defined using NSEG flat, straight lines.

A0,A1 (Real)

The angles, in degrees, defining the span of the section. Positive angles are measured counterclockwise from the positive X axis of the world coordinate system. Sections are created counterclockwise from A0 <u>to</u> A1.

4. <u>Discussion Example</u>

For complete 360 degree circles refer to the JCIRCL subroutine. For simple unfilled arc lines refer to the JARC subroutine.

Figure 2.5 is an example of a circle section created by the following subroutine call:

```
CALL JSECTR(1.0,2.0,0.0,3.0,1,10.0,50.0)
```

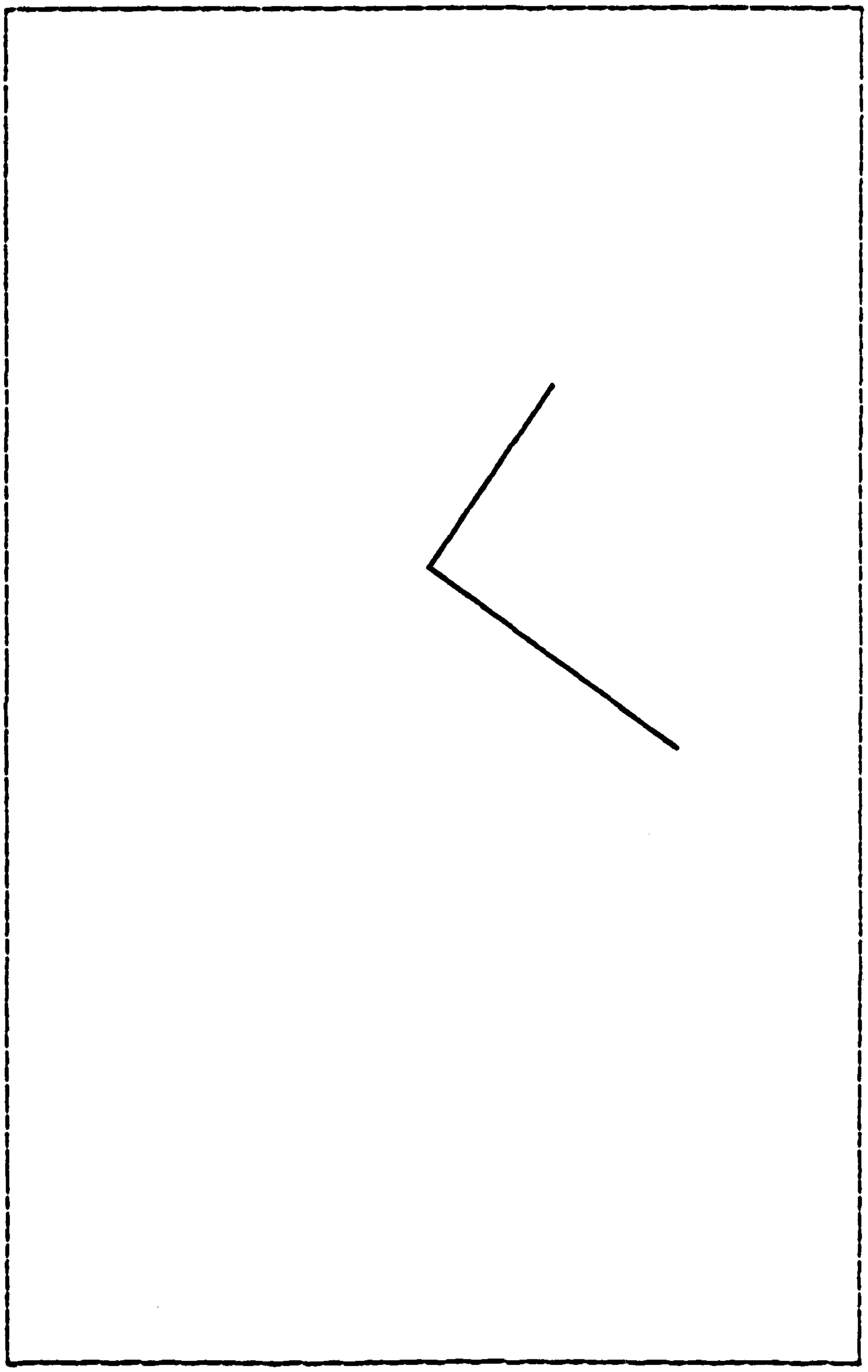


Figure 2.1 Broken Line.

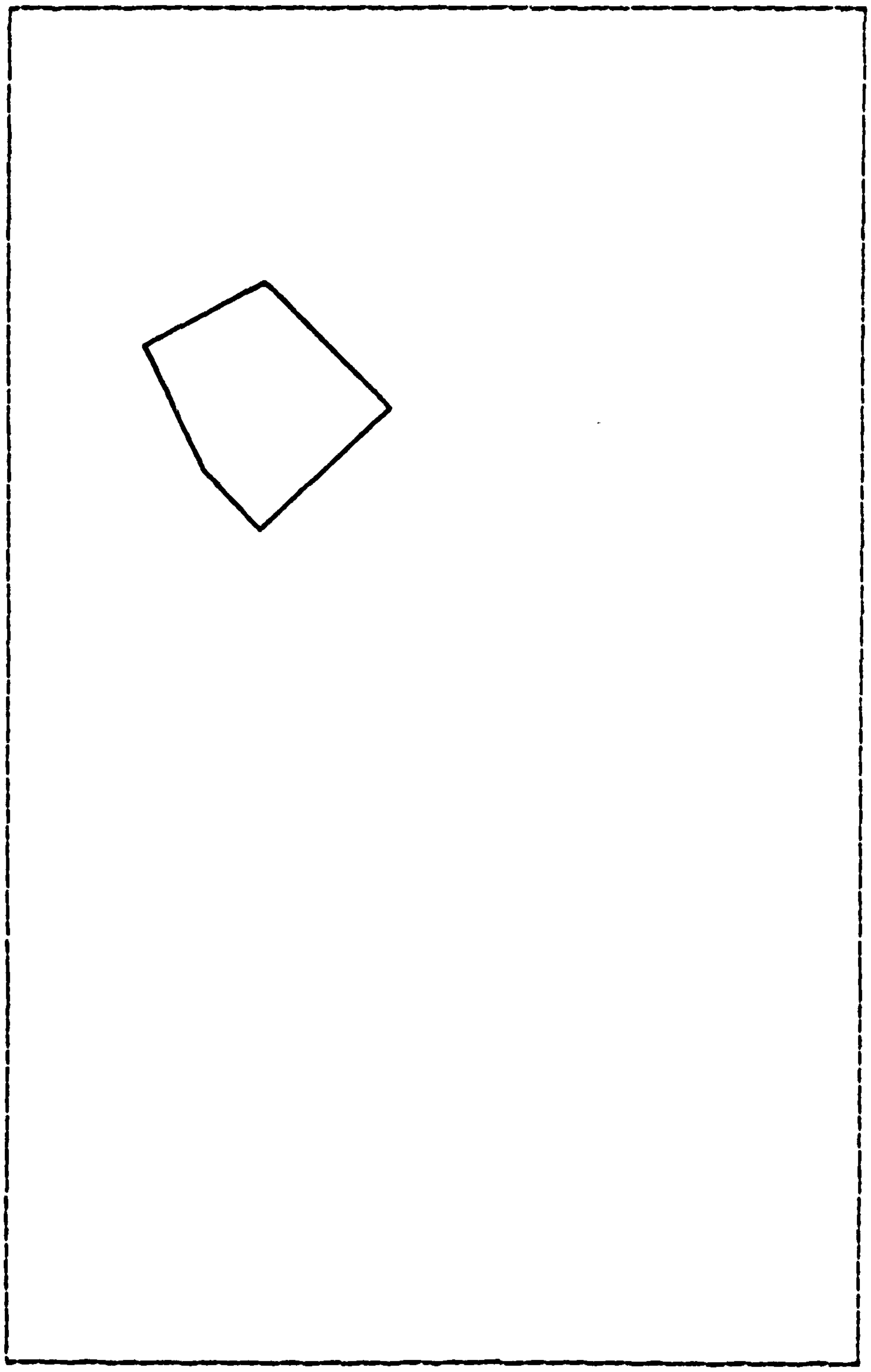


Figure 2.2 Five-Sided Polygon.

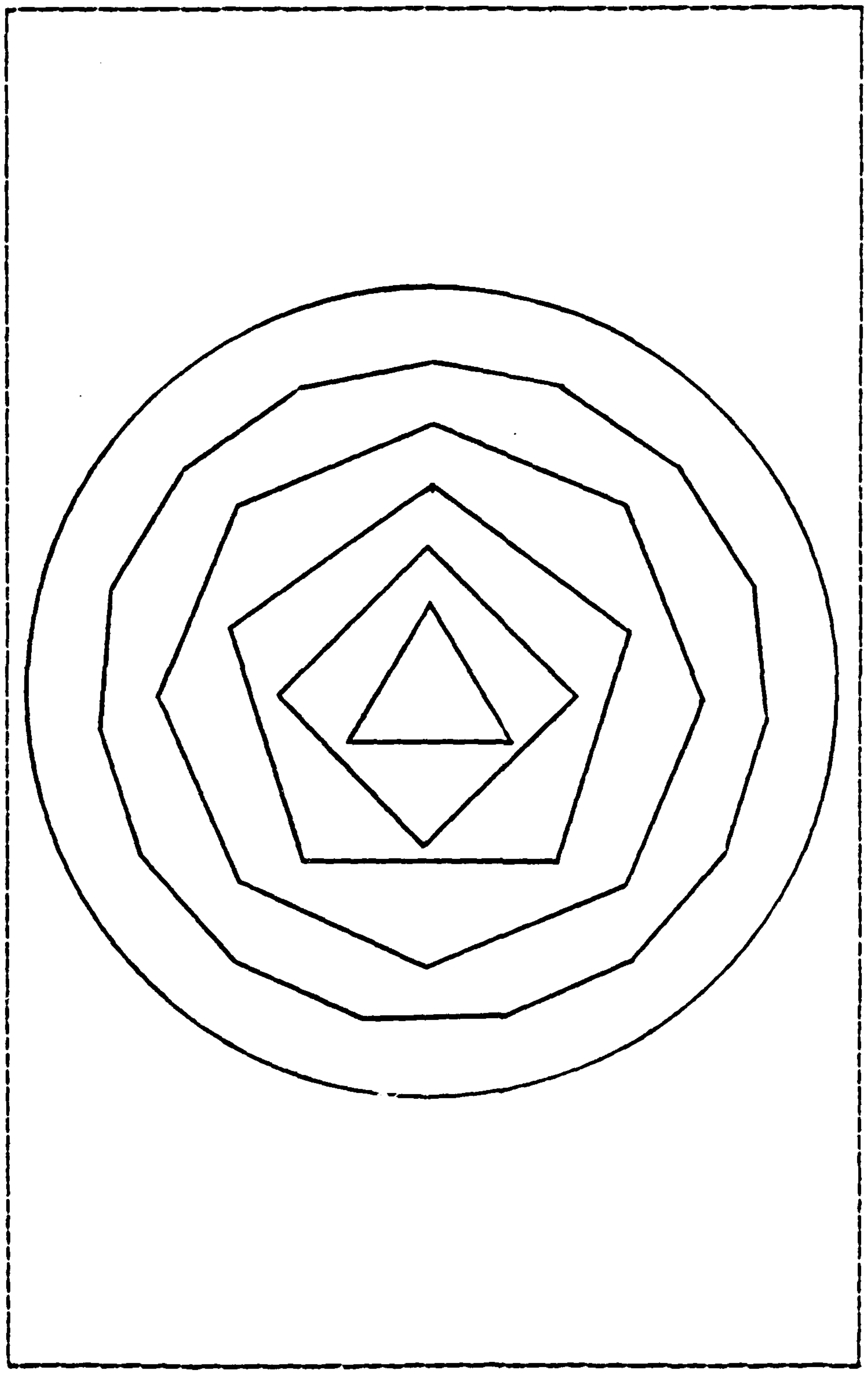


Figure 2.3 Circles.

```
C       .
C       .
C       .
         CALL  JCIRCL(0.,0.,0.,6.5,2)
         CALL  JCIRCL(0.,0.,0.,5.5,15)
         CALL  JCIRCL(0.,0.,0.,4.5,8)
         CALL  JCIRCL(0.,0.,0.,3.5,5)
         CALL  JCIRCL(0.,0.,0.,2.5,4)
         CALL  JCIRCL(0.,0.,0.,1.5,3)
C        .
C        .
C        .
```

Figure 2.4   Circle Creation Program.

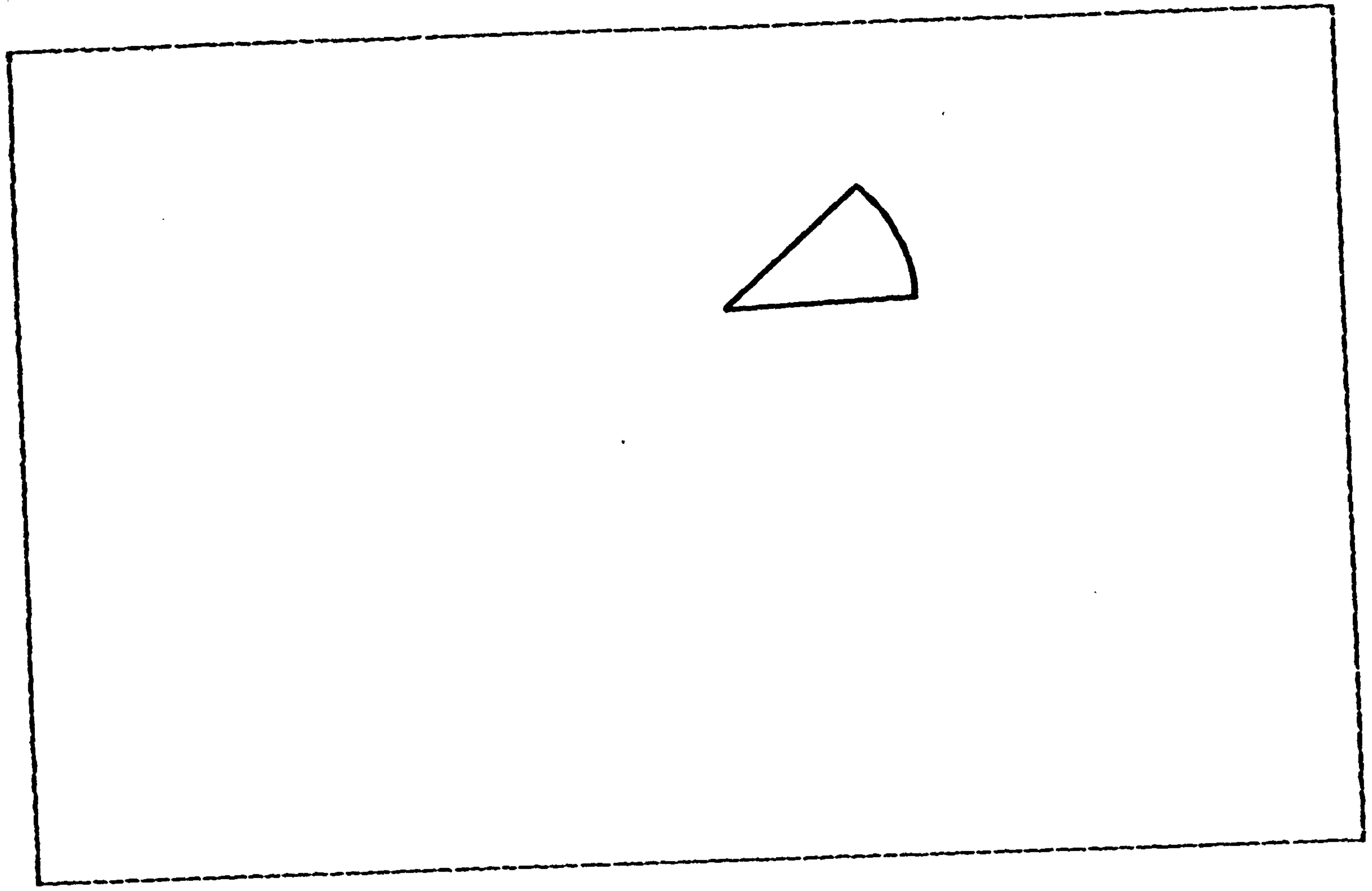


Figure 2.5 Circle Section.

III. ATTRIBUTES

A. GENERAL

Attributes define primitive image characteristics. Examples of attributes are "continuous" vs "dashed" lines, or even color itself. There are many different attributes possible, and they are all set using DI-3000 subroutine calls. The specifications described in this chapter will primarily apply to non-text primitives, but in several instances both text and non-text will share the same call. Text-only attributes are discussed in Chapter IV.

B. DEFAULT AND CURRENT ATTRIBUTE VALUES

Each primitive attribute has one of two values, a default value and a current value. The default value is set automatically when DI-3000 is initialized (with the required JBEGIN subroutine), and remains in effect unless changed before the first program segment has been opened.

A program segment is a program section and a graphics data structure. It contains a sequence of primitive calls that create a graphics sub-image of logically related objects, and is always used when creating images. Breaking down a complex picture into program segments simplifies the creation procedure and aids understanding. It is required in all DI-3000 programs. A detailed discussion of segmenting can be found in Chapter VI, but for our purposes the following brief example should suffice.

Assume a programmer wishes to draw a green square in the upper right-hand portion of the screen, and a red semicircle in the center. One method for accomplishing this would be to create a segment for each primitive image.

Let segment number one be for a square, and segment number two for a semicircle. As discussed in Chapter II we would use the appropriate subroutine calls for these image primitives within each created segment. But, so far we have not explained how to specify different image colors using DI-3000 attributes.

One method would be as follows. Set the default color to green, begin (open) the first segment, create and display the square, and end (close) the first segment. Then, change the default color to red, open the second segment, create and display the semicircle, and close the second segment. This method would not work because DI-3000 does not permit changing the default color once the first segment has been opened. Therefore, we would have to change the <u>current</u> attribute color "within" the second segment to red.

Any further segments would again be drawn in default color green, unless the current color attribute was changed within them. A program excerpt that creates our example is shown in Figure 3.1

Default values are usually set by the programmer to the most common image attribute. If a complex picture will be predominantly of one color, then that default color will be set. Only images of different colors will need to be specified later using the current color attribute calls within segments.

Most default attributes are automatically set to their most common values when a DI-3000 program is begun and are usually not changed. These initial, automatic settings are listed in the description of each attribute call.

Default and common attribute value subroutine calls can be distinguished from each other by their format. Default attributes are set using the form:

JDXXXX (e.g. JDCOLR)

Current attribute calls are of the form:

JXXXXX (e.g. JCOLOR)

Current attribute values can be changed as often as necessary within segments.

C. POLYLINE AND POLYGON ATTRIBUTE CLASSES

Non-text attributes apply to either polygon or polyline primitive images and are classified as such. As described in Chapter II, polygon images always result in the creation of an enclosed figure, while polyline images do not necessarily.

Throughout this chapter, a detailed description of the polyline class will first be given, followed by the polygon class. Attribute characteristics are listed by these two classes in the primitive descriptions in Chapter II. Therefore, a programmer can first refer to the discussion in Chapter II of the image to be created, then cross-reference to this chapter by attribute class for methods that will specify image characteristics.

D. POLYLINE COLOR (JCOLOR/JDCOLR)

1. Description

JCCLOR/JDCOLR are the subroutines used to set the current/default polyline primitive color attribute.

2. Use

```
CALL JCOLOR(CVALUE)    (Current)
                 or
CALL JDCOLR(DVALUE)    (Default)
```

3. Parameter Definition

CVALUE (Integer)

The color index of subsequent primitives within the currently open segment. $(0 \le CVALUE \le 8)$

DVALUE (Integer)

The new value for the default color index. When a segment is opened, the color index is set to DVALUE. (0≤DVALUE≤8)

4. Applicability

Applies to draws, polylines, polygon edges, and text primitives.

5. Initialization Default Value

DVALUE=0 (Background color)

6. Discussion

The following eight entries in the color lookup table are downloaded to the graphics processor by setting CVALUE/DVALUE to any of the following integer values:

0 --	Background (No color)
1 --	Red
2 --	Green
3 --	Yellow
4 --	Magenta (Dark Blue)
5 --	Purple
6 --	Cyan (Light Blue)
7 --	White
8 --	Background complement (White also)

JDCOLR at program beginning or JCOLOR within each segment must be specified. If not, no image will appear, because JDCOLR initialization default value is 0 and primitives will be drawn in background color.

The default entries in the color table are limited as described (at present). Ramtek is capable of creating many different colors (any hue, lightness, or saturation desired), but for simplicity in this tutorial a discussion of how to accomplish this is not included. For further information, refer to the Reference 1 description of subroutine JCOTBL (creating a color table). (0≤CVALUE/DVALUE≤32767) is the actual range of permissible values, but to utilize any value >8 your color table must first be defined. If this definition is not done, the colors called by values >8 will be unpredictable.

E. POLYLINE STYLE (JLSTYL/JDLSTY)

1. Description

JLSTYL/JDLSTY are the subroutines used to set the current/default polyline primitive line style attribute. Lines can be continuous or a combination of varying lengths of "dotted" or "dashed" portions.

2. Use

```
CALL JLSTYL(CVALUE)    (Current)
          or
CALL JDLSTY(DVALUE)    (Default)
```

3. Parameter Definition

CVALUE (Integer)

The line style of subsequent primitives within the currently open segment. (0≤CVALUE≤32767)

DVALUE (Integer)

The new value for the default line style. When a segment is opened, the line style is reset to DVALUE. (0≤DVALUE≤32767)

4. Applicability

Applies to polyline primitives only. Does not apply to polygon edges, or any text.

5. Initialization Default Value

DVALUE=0 (Solid line)

6. Discussion

The Ramtek provides a multitude of different line styles. As CVALUE/DVALUE values are increased the spacing between "dots" and "dashes" increases, as does their length and sequence. Every value that is a multiple of 8 will result in a solid line, with a new combination of styles to follow. It is best to experiment with available values to find the exact desired line style, but in general a value of CVALUE/DVALUE=7 gives a good "dotted" line that is easily distinguishable from normal "solid" lines.

F. POLYLINE INTENSITY (JINTEN/JDINTE)

1. Discussion

JINTEN/JDINTE do not alter line intensity as described in Reference 1. To vary intensity, a color table must be created.

G. POLYLINE WIDTH (JLWIDE/JDLWID)

1. Discussion

JLWIDE/JDLWID do not alter line width as described in Reference 1. To vary line width, the JWINDO command must be used to "blow up" or "shrink" image size by changing world coordinate size. This technique is explained in Chapter V.

H. POLYGON EDGE COLOR STYLE (JPEDGE/JDPEDG)

1. Description

JPEDGE/JDPEDG are the subroutines used to set the current/default polygon edge (border) color style. Edge color style is of two types: "same as" or "different than" polygon interior color.

2. Use

```
CALL JPEDGE(CVALUE)    (Current)
          or
CALL JDPEDG(DVALUE)    (Default)
```

3. Parameter Definition

CVALUE (Integer)

The polygon edge color style of subsequent polygons within the currently open segment. ($0 \le CVALUE \le 32767$)

DVALUE (Integer)

The new value for the default polygon edge color style. When a segment is opened, the edge style is reset to DVALUE. ($0 \le DVALUE \le 32767$)

4. Applicability

Applies only to polygon primitives.

5. Initialization Default Value

DVALUE=0 (Border Visible)

6. Discussion

Polygons are drawn with their borders either of the same color as their interiors, or of different colors. If the same color is specified, the border will be invisible. If not, the border will form an edge of different color around the polygon.

The following CVALUE/DVALUE values are used to specify edge style:

Odd -- Invisible border.

Even -- Border visible.

The polygon edge takes on the characteristics of current color when visible. Therefore, polyline attribute JCOLCR/JDCOLR determines polygon edge color in the "Even" style.

I. POLYGON INTERIOR STYLE (JPINTR/JDPINT)

1. Description

JPINTR/JDPINT are the subroutines used to set the current/default polygon interior style attribute. Interior style is either "empty" (no fill, background color), or "filled" (using an interior color as specified by the JPIDEX/JDPIDX subroutine calls).

2. Use

```
CALL JPINTR(CVALUE)    (Current)
           or
CALL JDPINT(DVALUE)    (Default)
```

3. Parameter Definitions

CVALUE (Integer)

The polygon interior style of subsequent polygons within the currently open segment. (CVALUE= 0 or 1)

DVALUE (Integer)

The new value for the default polygon interior style. When a segment is opened, the interior style is reset to DVALUE. (DVALUE= 0 or 1)

4. Applicability

Applies only to polygon primitives.

5. Initialization Default Value

DVALUE=0 (No polygon fill)

6. Discussion

Fully connected lines can be created by specifying the "no fill" polygon attribute. When polygon fill is desired, the color of the fill is determined by using the JPIDEX/JDPIDX subroutines as next described in this chapter. Note that JCOLCR/JDCOLR do not determine polygon interior color.

J. POLYGON INTERIOR COLOR (JPIDEX/JDPIDX)

1. Description

JPIDEX/JDPIDX are the subroutines used to set the current/default polygon interior color.

2. Use

```
CALL JPIDEX(CCOLOR,0)    (Current)
                  or
CALL JDPIDX(DCOLOR,0)    (Default)
```

3. Parameter Definitions

CCOLOR (Integer)

The polygon interior color of subsequent polygons within the currently open segment. $(0 \le CCOLOR \le 8)$

DCOLOR (Integer)

The new value for the default polygon interior color. When a segment is opened, the interior color is reset to DCOLOR. $(0 \le DCOLOR \le 8)$

0 (Integer number zero)

Polygon interior style cannot be specified as described in Reference 1. Therefore, a "0" is used here as a placeholder only. All interiors can be "solid" filled or empty. Hatching is not supported.

4. Applicability

Applies only to polygon primitives.

5. Initialization Default Value

DCOLOR=0 (No interior color)

6. Discussion

Polygon interior color applies only to polygons whose interiors have been specified as "filled" using the JPINTR/JDPINT subroutines with value one. Interior color is not specified using the JCOLOR/JDCOLR subroutines.

The following color index table is applicable to the Ramtek monitors by setting CCOLOR/DCOLOR to the integer values:

0	--	Background (No color)
1	--	Red
2	--	Green
3	--	Yellow
4	--	Magenta (Dark Blue)
5	--	Purple
6	--	Cyan (Light Blue)
7	--	White
8	--	Background complement (White also)

Additional colors may be specified by referring to the Reference 1 description of JCOTBL (creating a color table). ($0 \leq$CCOLOR/DCOLOR≤ 32767) is the actual range of permissible values, but to utilize values >8 your color table must first be defined. If this definition is not done, the colors called by values >8 will be unpredictable.

K. POLYGON PROGRAM EXAMPLE

Figure 3.2 is a complete program that draws a polygon with red interior and yellow border.

```
C     .
C     .
C     .
C AT THE BEGINNING OF THE PROGRAM (BEFORE THE FIRST
C SEGMENT IS OPENED) DEFAULT COLOR IS SET TO GREEN.
C     .
C     .
      CALL JDCOLR(2)
C     .
C     .
C OPEN SEGMENT ONE FOR THE GREEN SQUARE, CREATE IT,
C THEN CLOSE SEGMENT ONE.
C     .
C     .
      CALL JROPEN(1)
      CALL JMOVE(5.,4.)
      CALL JDRAW(6.,4.)
      CALL JDRAW(6.,5.)
      CALL JDRAW(5.,5.)
      CALL JDRAW(5.,4.)
      CALL JRCLOS
C     .
C     .
C OPEN SEGMENT TWO FOR THE RED SEMI-CIRCLE.
C NOTE THAT CURRENT COLOR ATTRIBUTE IS CHANGED
C WITHIN SEGMENT TWO TO RED.  CREATE THE RED
C CIRCLE, THEN CLOSE SEGMENT TWO.
C     .
C     .
      CALL JROPEN(2)
      CALL JCOLOR(1)
      CALL JARC(0.,0.,0.,2.,0,0.,180.)
      CALL JRCLOS
C     .
C     .
C     .
```

Figure 3.1 Color Segment Program.

```
C THIS PROGRAM WILL CREATE A FIVE-SIDED POLYGON AND
C CISPLAY IT WITH A RED INTERIOR AND YELLOW EDGE.
C     .
C INITIALIZE THE PROGRAM.  DECLARE VARIABLES, AND
C ASSIGN VALUES TC NECESSARY ARRAYS.
C     .
      INTEGER MON
      REAL POSITX(5),POSITY(5)
      DATA POSITX /5.,2.5,-2.5,-6.5,7./
      DATA POSITY /6.,0.,-6.5,5.,7.5/
C     .
C INITIALIZE THE RAMTEK MONITOR AND COMMENCE MANDATORY
C DI-3000 INITIALIZATION SUBROUTINES.
C     .
      TYPE *, 'ENTER THE MCNITOR NUMBER'
      ACCEPT *, MCN
      CALL JBEGIN
      CALL JFILES(3,1,MON)
      CALL JDINIT(1)
      CALL JDEVON(1)
      CALL JDCOLR(3)    !SET DEFAULT COLOR YELLOW FOR ALL
                        !POLYLINES AND THE POLYGON EDGE.
      CALL JVSPAC(-1.,1.,-.8,.8)
      CALL JWINDO(-10.,10.,-8.,8.)
C     .
C OPEN SEGMENT ONE AND DRAW THE POLYGON.
C     .
      CALL JROPEN(1)
         CALL JPIDEX(1,0)   !SET INTERIOR COLOR TO RED.
         CALL JPINTR(1)     !FILL VICE NO-FILL.
         CALL JPEDGE(2)     !SET EDGE STYLE EVEN SO
                            !IT WILL BE VISIBLE AND WILL
                            !DRAW USING JDCOLOR DEFAULT
                            !COLOR YELLOW.
         CALL JPOLGN(POSITX,POSITY,5)  !CREATE AND DRAW
                                       !THE POLYGON.
      CALL JRCLOS    !CLOSE SEGMENT ONE.
C     .
C     .
C PAUSE THE PROGRAM SO THE IMAGE WILL REMAIN ON THE
C SCREEN, THEN ENC THE PROGRAM.
C     .
C     .
      CALL JPAUSE(1)
      CALL JEND
      END
C     .
```

Figure 3.2   Typical Polygon Creation Program.

IV. TEXT PRIMITIVES AND ATTRIBUTES

A. GENERAL

Written text can be displayed on the Ramtek monitors. Subroutine calls exist that create these character primitives and control text attributes such as orientation, size, and type.

DI-3000 supports four different levels of text precision. This tutorial will describe the highest level, known as graphic arts precision text. Each character in a graphic arts string is "stroke generated" by DI-3000 software, resulting in the highest possible quality of text, rather than using any hardware character generator. Learning how to create graphic arts text is no more difficult than learning how to create the lower quality types because all text attributes apply universally.

If a programmer is concerned with transmitting a graphics program over low bandwidth communications lines, then high quality text creation may be excessively slow. In this case, refer to Reference 1 for a description of lower level text using the J1TEXT, J2TEXT, and J3TEXT commands.

Text primitives are only defined at absolute positions. There are no corresponding relative subroutine primitive calls.

B. TEXT ATTRIBUTES

As with non-text attributes, text attributes have either default or current values. Default values must be specified prior to the opening of the first segment, and current values can only be changed while a segment is open (within a segment). For a more detailed discussion refer to Chapter III.

All text attributes apply only to text primitives. One non-text attribute applies to text: the polyline color command (JCOLOR/JDCCLR). A description of these color subroutine calls can be found in Chapter III.

All text attributes have initialization default values. In all but one case, these values are such that text attributes need not be changed because legible, normal characters are output. Only the character size attribute (JSIZE/JDSIZE) must be specified. Without its modification, the text will be much too small to be readable since size is specified in terms of world coordinate window. The 16.0 by 20.0 convention chosen for the examples in this tutorial makes this size change necessary.

C. CREATING A CHARACTER STRING (JHTEXT)

1. Description

JHTEXT is the subroutine used to output a graphic arts quality text string.

2. Use

```
CALL JHTEXT(NCHARS,STRING)
```

3. Parameter Definition

NCHARS (Integer)

The number of characters in the text string. (0<=NCHARS<=255).

STRING (Integer, Hollerith Input String)

The actual, literal character string to be output. For example, a STRING value of 18HTHIS IS AN EXAMPLE would output THIS IS AN EXAMPLE on the monitor screen.

Note -- STRING must be an Integer type variable. If your program uses CHARACTER* variable types, they must be converted to integer before using them as parameters in this JHTEXT subroutine call.

STRING can also contain "sentinel" characters as discussed below for changing to upper or lower case, underlining, or other special functions.

An example of the complete JHTEXT subroutine call that would output the letters THIS IS AN EXAMPLE follows:

```
CALL JHTEXT(18,18HTHIS IS AN EXAMPLE)
```

4. Required Attributes

All text class attributes and the non-text polyline color attribute JCOLOR/JDCOLR apply.

5. Discussion Example

Characters are drawn in current polyline color as defined either by JCOLOR or JDCOLR. Refer to Chapter III for a description of these attributes.

If increased space between, or overlap of, characters is desired refer to the gap attribute (JGAP/JDGAP).

Simple block letters are created with the initial default. If more artistic styles are desired, refer to the font attribute (JFONT/JDFONT).

Initial character size will be too small to be legible using the normal size attribute default and the window of 16.0 by 20.0 chosen for this tutorial. Therefore, the JSIZE or JDSIZE subroutines _must_ be used prior to writing any text unless very small world coordinate window size is chosen.

Character string direction will be "left-to-right" unless specified differently by changing the path attribute (JPATH/JDPATH). Various angle orientations can also be specified by using the character base (JBASE/JDBASE) and character plane (JPLANE/JDPLAN) attributes discussed in Reference 1.

Character justification will begin with the "lower-left-hand" corner of the string corresponding to the current position (CP) on the screen. Any changes to justification can be made by changing the (JJUST/JDJUST) attributes.

All text primitive subroutine calls return CP to its initial position when the call completes.

Sentinel characters are allowed within the STRING parameter to specify different functions. All sentinel character functions are prefaced by the start command (open bracket, '[') and terminated by the end command (closed bracket, ']'). The sentinel character functions are:

[BSUP]	-- Begin superscript
[ESUP]	-- End superscript
[BSUB]	-- Begin subscript
[ESUB]	-- End subscript
[BUC]	-- Begin uppercase
[EUC]	-- End uppercase
[BLC]	-- Begin lowercase
[ELC]	-- End lowercase
[BUND]	-- Begin underline
[EUND]	-- End underline
[FONT=n]	-- Change to font number 'n'

Figure 4.1 shows examples of sentinels and their resulting outputs.

D. CHARACTER SIZE (JSIZE/JDSIZE)

1. Description

JSIZE/JDSIZE are the subroutines that set the current/default text primitive character size.

2. Use

```
CALL JSIZE(CXSIZE,CYSIZE)    (Current)
                  or
CALL JDSIZE(DXSIZE,DYSIZE)    (Default)
```

3. Parameter Definition

CXSIZE,CYSIZE (Real)

The size of a character within the currently open segment.

DXSIZE,DYSIZE (Real)

The new value for the default size of a character. When a segment is opened, character size is reset to DXSIZE,DYSIZE.

CXSIZE/DXSIZE are widths (in the X-axis direction) and CYSIZE/DYSIZE are heights (in the Y-axis direction). They are expressed as world coordinate values.

4. Initialization Default Value

DXSIZE/DYSIZE = 0.02

5. Discussion

JSIZE/JDSIZE parameters are expressed as world coordinates. Therefore, since JWINDO specifies the world coordinate grid there is a direct relationship between window and character size. Both dimensions will be defined in terms of the same units. Polyline and polygon primitives also use these world coordinate units.

E. CHARACTER SPACING (JGAP/JDGAP)

1. Description

JGAP/JDGAP are the subroutines that set the current/default text spacing (gap) between characters.

2. Use

CALL JGAP(CVALUE) (Current)

or

CALL JDGAP(DVALUE) (Default)

3. Parameter Definitions

CVALUE (Real)

The spacing between characters within the currently open segment. (CVALUE > -1.0)

DVALUE (Real)

The new value for the default spacing between characters. When a segment is opened, spacing is reset to DVALUE. (DVALUE > -1.0)

4. Initialization Default Value

DVALUE = 0.0 (Normal spacing)

5. Discussion

The spacing between characters is defined as a multiple of character width. Therefore, if CVALUE/DVALUE = 1.0 there will be a space between each character box equivalent to the width of a normal character (for example, an 'N').

Notice that when CVALUE/DVALUE = 0.0 there is still some space between characters. This is because each character is surrounded by a character box that contains normal pad space so that text does not "run together."

Text can be made to partially or completely overlap by setting CVALUE/DVALUE < -.25. CVALUE/DVALUE = -.25 makes the characters "touch" each other, and CVALUE/DVALUE = -1.0 superimposes all characters onto a single location.

F. CHARACTER STRING DIRECTION (JPATH/JDPATH)

1. Description

JPATH/JDPATH are the subroutines that set the current/default text primitive character direction (path). Normal direction is "left-to-right."

2. Use

CALL JPATH(CVALUE) (Current)

or

CALL JDPATH(DVALUE) (Default)

3. Parameter Definition

CVALUE (Integer)

The direction of a character string within the currently open segment. (CVALUE=1 thru 4)

DVALUE (Integer)

The new default value for character string direction. When a segment is opened, direction is reset to DVALUE. (DVALUE=1 thru 4)

4. Initialization Default Value

DVALUE=1 (left-to-right)

5. Discussion

Character strings can be generated in four different directions using the JPATH/JDPATH subroutines by setting their parameters to the following values:

1 --	left-to-right	(character path right)
2 --	top-to-bottom	(path down)
3 --	right-to-left	(character path left)
4 --	bottom-to-top	(path up)

Values 3 and 4 will result in inverted characters unless used in conjunction with base and plane manipulation as discussed in Reference 1. Normal directions can be generated using values 1 or 2.

Figure 4.2 is an example of the four possible path settings.

G. CHARACTER STRING JUSTIFICATION (JJUST/JDJUST)

1. Description

JJUST/JDJUST are the subroutines that set the current/default text primitive character string justification. The justification point is the starting position of a character string. It lies "within" the string (for example, the "lower-left-hand corner," or the "center" of the string). Figure 4.3 shows examples of string justification.

2. Use

```
CALL JJUST(CHORIZ,CVERT)    (Current)
                or
CALL JDJUSI(DHORIZ,DVERT)   (Default)
```

3. Parameter Definition

CHCRIZ/CVERT (Integer)

The horizontal/vertical justification of a character string within a currently open segment. (CHORIZ,CVERT=1,2, or 3)

DHCRIZ/DVERT (Integer)

The new default value for the horizontal/vertical character string justification. When a segment is opened, justification is reset to DHORIZ and DVERT. (DHORIZ,DVERT=1,2, or 3)

4. Initializaticn Default Value

DHORIZ,DVERT=1 (bottom left)

5. Discussion

Character string justification values are defined as follows:

CHORIZ/DHORIZ	CVERT/DVERT
1 -- left	1 -- bottom
2 -- center	2 -- center

3 -- right   3 -- top

The justification point refers to the position indication "dot" shown on the examples in Figure 4.3. When a text primitive string is created, this point is overlaid (mapped) onto current screen position (CP), thus providing for nine different orientations.

H. CHARACTER STYLE (JFONT/JDFONT)

1. Description

JFONT/JDFONT are the subroutines that set the current/default character style (font).

2. Use

```
CALL JFONT(CVALUE)    (Current)
          or
CALL JDFONT(DVALUE)   (Default)
```

3. Parameter Definition

CVALUE (Integer)

The style type of a character string within a currently open segment. (1<=CVALUE<=12)

DVALUE (Integer)

The new default value for character style type. When a segment is opened, style type is reset to DVALUE. (1<=DVALUE<=12)

4. Initialization Default Value

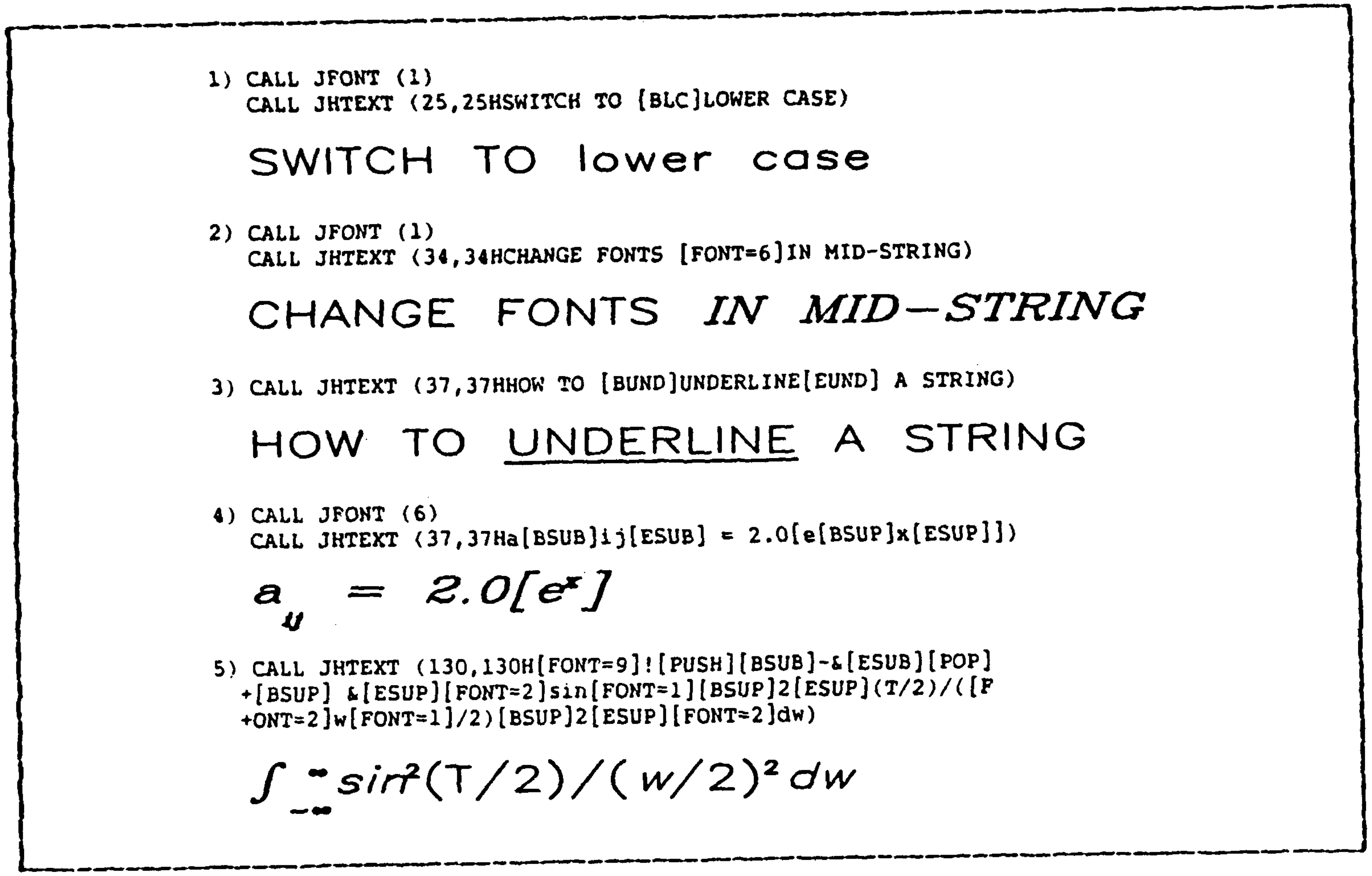
DVALUE=1 (simplex block)

5. Discussion

There are six different character style fonts available. Each can be generated in block or italics (right-slanted) orientation. The following CVALUE/DVALUE values are used to specify style:

1 -- simplex block	7 -- triplex block
2 -- simplex italics	8 -- triplex italics
3 -- duplex block	9 -- Greek block
4 -- duplex italics	10 -- Greek italics
5 -- complex block	11 -- script block
6 -- complex italics	12 -- script italics

Figure 4.4 gives examples of the six basic character style fonts.



```
1) CALL JFONT (1)
   CALL JHTEXT (25,25HSWITCH TO [BLC]LOWER CASE)
```

SWITCH TO lower case

```
2) CALL JFONT (1)
   CALL JHTEXT (34,34HCHANGE FONTS [FONT=6]IN MID-STRING)
```

CHANGE FONTS *IN MID–STRING*

```
3) CALL JHTEXT (37,37HHOW TO [BUND]UNDERLINE[EUND] A STRING)
```

HOW TO <u>UNDERLINE</u> A STRING

```
4) CALL JFONT (6)
   CALL JHTEXT (37,37Ha[BSUB]ij[ESUB] = 2.0[e[BSUP]x[ESUP]])
```

$$a_{ij} = 2.0[e^{x}]$$

```
5) CALL JHTEXT (130,130H[FONT=9]![PUSH][BSUB]-&[ESUB][POP]
  +[BSUP] &[ESUP][FONT=2]sin[FONT=1][BSUP]2[ESUP](T/2)/([F
  +ONT=2]w[FONT=1]/2)[BSUP]2[ESUP][FONT=2]dw)
```

$$\int_{-\infty}^{\infty} \sin^2(T/2)/(w/2)^2\,dw$$

Figure 4.1 Sentinel Statements and Output Examples.

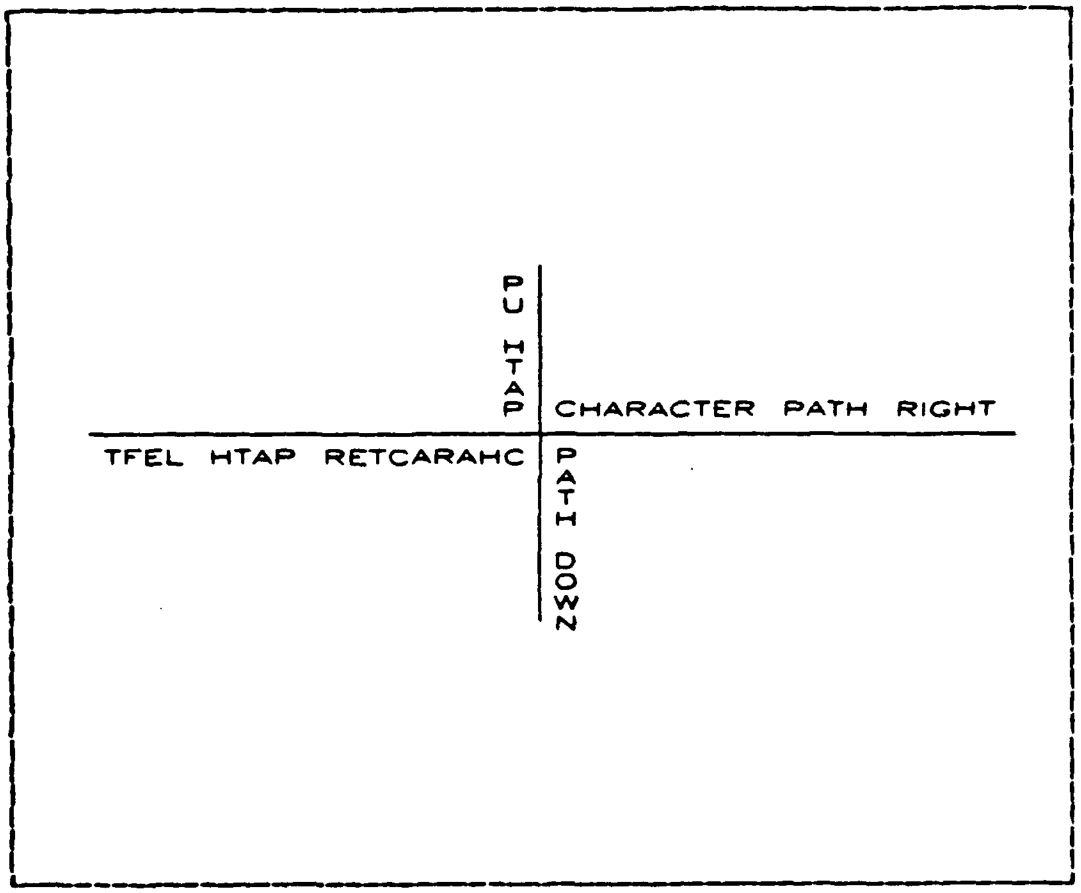


Figure 4.2 Character String Direction Path Example.

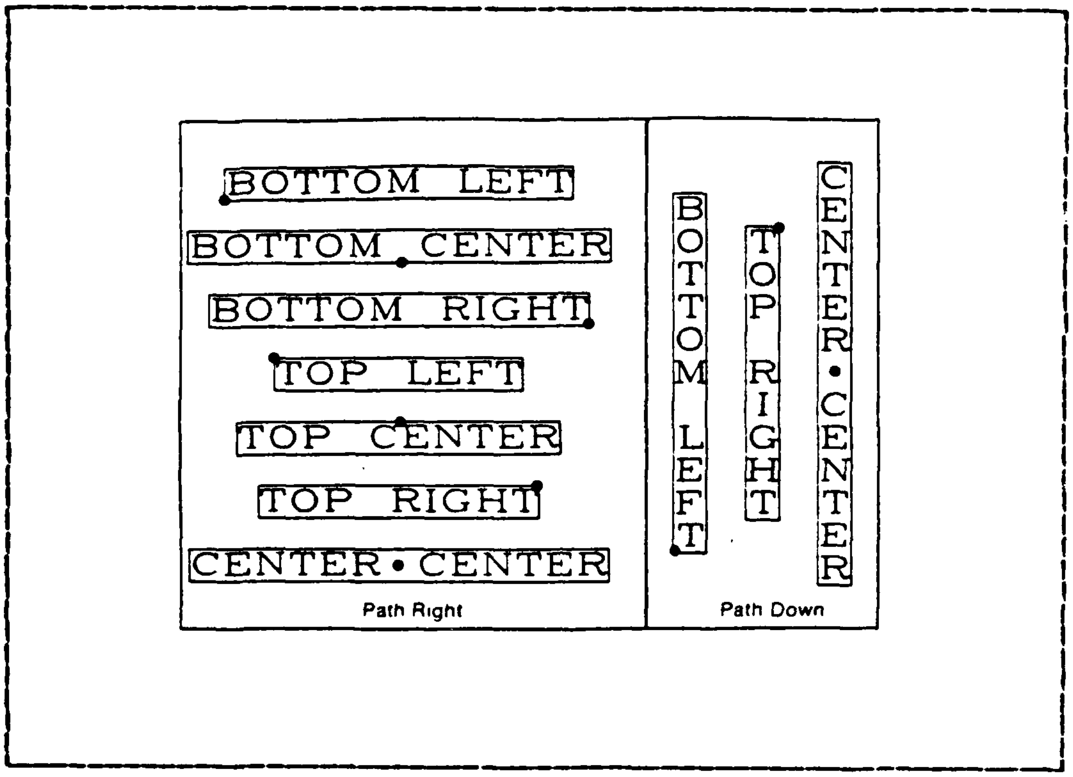
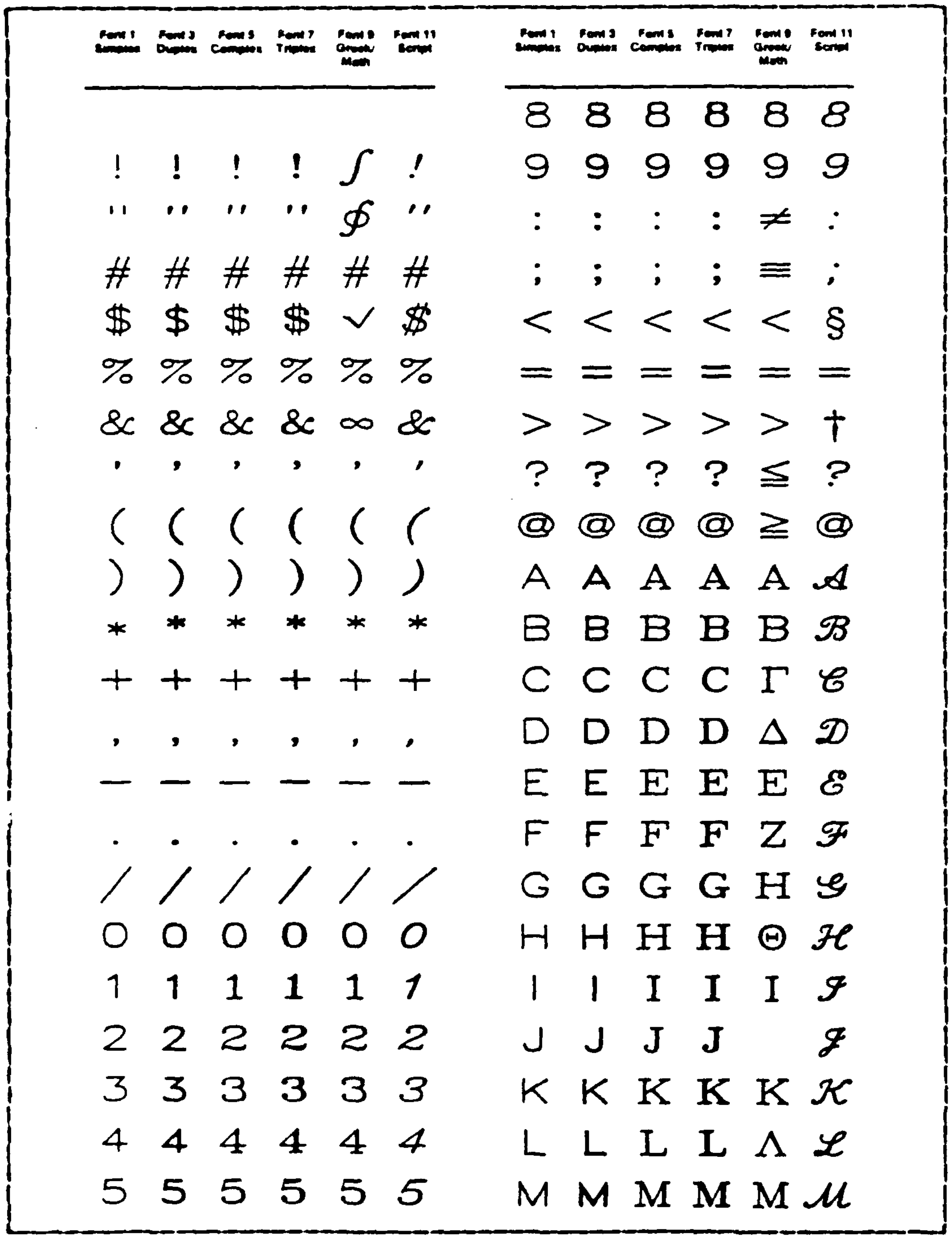


Figure 4.3   Text Justification Examples.



Font 1 Simplex	Font 3 Duplex	Font 5 Complex	Font 7 Triplex	Font 9 Greek/ Math	Font 11 Script	Font 1 Simplex	Font 3 Duplex	Font 5 Complex	Font 7 Triplex	Font 9 Greek/ Math	Font 11 Script
						8	8	8	8	8	8
!	!	!	!	∫	!	9	9	9	9	9	9
"	"	"	"	∮	"	:	:	:	:	≠	:
#	#	#	#	#	#	;	;	;	;	≡	;
$	$	$	$	√	$	<	<	<	<	<	§
%	%	%	%	%	%	=	=	=	=	=	=
&	&	&	&	∞	&	>	>	>	>	>	†
'	'	'	'	'	'	?	?	?	?	≤	?
(	(	(	(	(	(	@	@	@	@	≥	@
)	)	)	)	)	)	A	A	A	A	A	A
*	*	*	*	*	*	B	B	B	B	B	B
+	+	+	+	+	+	C	C	C	C	Γ	C
,	,	,	,	,	,	D	D	D	D	Δ	D
—	—	—	—	—	—	E	E	E	E	E	E
.	.	.	.	.	.	F	F	F	F	Z	F
/	/	/	/	/	/	G	G	G	G	H	G
0	0	0	0	0	0	H	H	H	H	Θ	H
1	1	1	1	1	1	I	I	I	I	I	I
2	2	2	2	2	2	J	J	J	J		J
3	3	3	3	3	3	K	K	K	K	K	K
4	4	4	4	4	4	L	L	L	L	Λ	L
5	5	5	5	5	5	M	M	M	M	M	M

Figure 4.4 Graphic Precision Text Styles.

V. REQUIRED SUBROUTINES

A. GENERAL

Figure 1.1 is an example of the minimum set of subroutine calls necessary to run any DI-3000 graphics program. The program harness is not meant to be an exhaustive list of all possible control commands, but it does provide for all basic initialization, primitive creation, primitive visibility, and termination requirements.

The remainder of this chapter will describe the commands listed and alternative subroutine calls if they apply. Some terminclogy already defined will be used.

B. INITIALIZATION (JBEGIN)

1. Description

JBEGIN is the subroutine call used to begin the DI-3000 graphics portion of an application program. It sets all default parameter and attribute values to their initialization state.

2. Use

```
CALL JBEGIN    (No parameters)
```

3. Discussion

Any Fortran statements in the application program may precede or follow the JBEGIN call if they adhere to the standard, required order of Fortran statements and lines as described in Reference 2. But, JBEGIN must be the first DI-3000 statement. It begins the application program graphics section.

JBEGIN does not specify an output device. The JFILES, JDINIT, and JDEVON commands are used after JBEGIN to define, initialize, and select the Ramtek monitor pair to be used for display.

C. SELECTING AND OPERATING RAMTEK MONITORS

1. Discussion

The war lab has three graphics processors and two Ramtek monitors for each processor. Before trying to use one of the available units the programmer must verify that at least one monitor pair is not already in use. There are three pairs currently accessible. Figure 1.1 provides a typical monitor configuration. Each monitor can be identified as belonging to monitor pair 1, 2, or 3 by the label beneath the screen. If you try to access a pair in use your program will run-terminate with an error.

A program could be written that would only access a single monitor pair, but this would limit the user. To make all programs able to use all monitors, the following two Fortran statements are included in the Figure 1.3 program harness and must precede the JBEGIN call in any application program:

```
TYPE *, 'ENTER THE MONITOR NUMBER'
ACCEPT *, MON
```

Fortran programmers may be unfamiliar with these statements because they are VAX-11 Fortran extensions to the Fortran-77 standard. TYPE merely queries the user at the terminal (interactively during program run) as to which monitor pair is intended for use. ACCEPT inputs the monitor number and stores it in an integer variable location named MON for later use in the JFILES subroutine call.

Both screens of a monitor pair do not need to be energized unless you wish to display two identical pictures simultaneously. Before running your program, insure that the two Ramtek front panel toggle switches are in the "up" positions and that the brightness knob is turned fully counterclockwise. There is a one minute warm-up time required if the screens were previously deenergized.

D. DEFINING MONITOR UNIT PAIR NUMBER (JFILES)

1. Description

JFILES is the subroutine call that determines which Ramtek monitor pair will display the graphics image.

2. Use

```
CALL JFILES(CODE,1,MONNUM)
```

3. Parameter Definition

CODE (Integer)

A code that indicates which DI-3000 internal file will be overridden. Normally CODE=3. If graphics input is to be done using the input tablets, CODE=4 is required.

1 (Integer number one)

Required because of current file specification as implemented on the VAX operating system in the War Lab.

MONNUM (Integer)

The monitor pair number to be used to display the application program graphics picture. (MONNUM= 1,2, or 3)

4. Discussion

MONNUM can be any variable name used by the application program to indicate monitor pair number. If the program harness of Figure 1.3 is used, then MONNUM must be the variable named MON:

```
CALL JFILES(3,1,MON)
```

If a user knows that a certain screen pair will always be used, an integer value of 1, 2, or 3 can instead be used as MONNUM value. For example, if screen pair 2 were always tc be used the following statement would apply:

```
CALL JFILES(3,1,2)
```

E. INITIALIZATION AND SELECTION (JDINIT/JDEVON)

1. Description

JDINIT and JDEVON are the subroutine calls that initialize and then select, respectively, display devices. They must be included in the mandatory sequence of DI-3000 subroutine calls.

2. Use

```
CALL JDINIT(MON)    (Must be in this order)
CALL JDEVON(MON)
```

3. Parameter Definition

MON (Integer)

The current version of DI-3000 assigns device number one to all Ramtek display monitors. This number must be used as the parameter value of both subroutine calls (MON=1).

4. Discussion

JDINIT(MON) and JDEVON(MON) must precede any default attribute declaration subroutines or any segments.

F. SPECIFYING COLOR

1. Discussion

A color must be specified if any polylines or text are to be visible. Either the JDCOLR subroutine prior to opening the first segment, or the JCOLOR subroutine within each segment can be used.

If polygons are to be created, use the polygon color attributes JPIDEX/JDPIDX in conjunction with the interior style attributes JPINTR/JDPINT.

In general, for the beginning user it is a safe practice to always specify a polyline default color prior to opening the first segment. This is done in the Figure 1.3 program harness example with the JDCOLR(2) subroutine call. Since color value 2 is used, polylines and text would be drawn in green.

G. DEFINING COORDINATE ASPECT RATIO (JVSPAC)

1. Description

JVSPAC is the subroutine call that defines the actual Ramtek monitor screen area to be used.

2. Use

```
CALL JVSPAC(-1.0,1.0,-.8,.8)
```

3. Discussion

Real number parameters -1.0, 1.0, -.8, and .8 should be used where indicated if the maximum screen area available is to be utilized. Ramtek GM859C color monitors have a defined virtual coordinate display area ±1.0 unit wide by ±.8 units high, referenced to an invisible screen center point at (0.,0.). Other values would not fully utilize the available screen surface.

Portions of the screen can be used by varying JVSPAC parameters. The height-to-width ratio (.8) must be identical for these parameters and the JWINDO parameters if no distortion is to occur during the "mapping" of world coordinate picture onto virtual coordinate space.

For a further description of JVSPAC, see Reference 1.

H. DEFINING WORLD COORDINATE WINDOW (JWINDO)

1. Description

JWINDO is the subroutine call that defines the world coordinate system. All text and image points in a program are referenced to this grid.

2. Use

```
CALL JWINDO(XMIN,XMAX,YMIN,YMAX)
```

3. Parameter Definition

XMIN,XMAX (Real)

The minimum and maximum world coordinate boundaries in the horizontal (X-axis) direction.

YMIN,YMAX (Real)

The minimum and maximum world coordinate boundaries in the vertical (Y-axis) direction.

4. Discussion

The world cocrdinate system of this tutorial is an area 16.0 units high by 20.0 units across, chosen arbitrarily. The following command specifies this system:

```
CALL JWINDO(-10.0,10.0,-8.0,8.0)
```

Any real values could have been selected, but they would have to match the .8 height-to-width ratio, specified by the JVSPAC subroutine call, for no distortion to occur.

JWINDO can be used to "blow up" or "shrink" images merely by changing its parameter values. For example, if a user wants to double the size of an image, then the JWINDO parameters should be halved. This is analagous to an object existing in some world system, then finding itself in a world half the previous size. The object's size would seem to have doubled.

Distortions can also be created by changing height-to-width ratios to values other than .8.

JWINDO cannot be called while a segment is open.

I. SEGMENT REQUIREMENTS

1. Discussion

The JROPEN(1) and JRCLOS statements in Figure 1.3 begin and end a retained program segment. All images must be created within a segment. For images that can all appear on the screen at once, the program harness segment statements given will be adequate. Segments, and their visibility, are discussed in detail in Chapter VI.

J. ENDING A GRAPHICS PROGRAM (JEND/END)

1. Description

JEND is the subroutine call that terminates the DI-3000 graphics portion of an application program.

2. Use

```
CALL JEND   (No parameters)
```

3. Discussion

JEND must be the last DI-3000 statement in an application program. Any Fortran statements may precede or follow JEND if they adhere to the standard required order of Fortran statements and lines as discussed in Reference 2.

JEND insures that the Ramtek monitors previously initialized and selected are de-selected and terminated.

The END command in Figure 1.3 is a standard Fortran statement, and must be the last statement of any application program.

VI. DISPLAYING AN IMAGE

A. GENERAL

In addition to defining the shape and position of a primitive, a user can control when a DI-3000 picture will be displayed during program run. Also, simple images can be combined to create complex figures, and any combination of statements can be grouped together to form a program segment that can be made visible or invisible as a unit.

Normal execution of an application program will cause the DI-3000 graphics image to appear on the selected monitor pair, but the picture will clear immediately when the job ends. A description of how to "pause" a program so a display can be held for extended viewing is included.

This chapter explains how to control these functions, and discusses some image overlay limitations associated with the War Lab graphics system as configured.

B. SEGMENTS

A segment defines part of a whole picture of logically related objects (a graphics data structure). A complete graphics image normally consists of a sequence of segments.

Each segment is a series of DI-3000 primitive and current attribute calls. Every statement that creates a primitive must be contained within a segment. Only default attribute and initialization subroutines can exist outside this data structure.

An entire image creation sequence can be contained within a single segment, but multiple segments are often used to partition a program into more easily understandable parts. This technique also aids in error diagnosis.

There are two segment types: temporary and retained. Temporary segments are only displayed once. If they are ever cleared from the screen they cannot be restored. Retained segments are "named" segments that can be made visible and invisible as often as desired. In general, the retained segment offers the programmer more control over the picture and is the recommended type. A retained segment is used in the example program harness of Figure 1.3 in Chapter I.

When any segment is opened, the current position (CP) is set to zero (0.,0.) and all attributes are reset to their default values.

C. IMAGE OVERLAY LIMITATIONS

The War Lab graphics system utilizes certain conventions that can be limiting if compound images are desired. For example, if two or more primitives occupy the same pixel position on a screen the resultant image color will be a blend of the individual colors. This problem is not easily overcome, but sometimes can be corrected as follows. After drawing the second image over all or part of the first, the original image is made invisible by using the JVISBL command. When this is done, only the overlaid portion of the first image will disappear. A complete display of true color will result for both, but this technique can become very complicated for multiple images.

A second method involves accessing the Ramtek pixel data and conducting a bit-plane-erase as described in Reference 1 under Escape Functions. Trying to "black out" color by drawing with background color "0" does not work because you will actually be drawing with "nothing."

Polygon interiors are filled with color until they encounter a border, but this does not have to be the polygon's own border. If another polygon edge is encountered during the fill of an overlaying polygon, the fill will often stop. Again, using the JVISBL command to make the original, overlaid polygon invisible will often correct this problem because the limiting border will be removed.

When polygons are created, their defining points can cause their edge lines to intersect if the point sequence is incorrectly specified. This intersection will actually create more than one polygon. When color fill is attempted it will stop at the first intersection point rather than completing the interior color of the whole shape. Defining the shapes as multiple polygons corrects this problem.

D. CREATING A TEMPORARY SEGMENT (JOPEN/JCLOSE)

1. Discussion

JOPEN/JCLOSE are the subroutine calls that begin/end a temporary segment.

2. Use

```
CALL JOPEN
(sequence of DI-3000 subroutine calls)
CALL JCLOSE
```

3. Discussion

Temporary segments exist only once. Refer to JROPEN/JRCLOS for a discussion of retained segments.

Refer to Section B of this chapter for a description of segments in general.

E. CREATING A RETAINED SEGMENT (JROPEN/JRCLOS)

1. Description

JROPEN/JRCLOS are the subroutine calls that begin/end a retained segment.

2. Use

CALL JROPEN(NAME)

(Sequence of DI-3000 subroutine calls)

CALL JRCLOS

3. Parameter Definition

NAME (Integer)

The name of the retained segment to be opened. $(1 \le NAME \le 32000)$

4. Discussion

Retained segments can be made visible or invisible using the JVISBL/JDVSEL subroutine calls.

All retained segments can be cleared from the screen using the JFRAME and JVISBL subroutine calls together.

Retained segments can be erased from memory using the JCLEAR subroutine call.

Refer to Section B of this chapter for a description of segments in general.

Note -- JRCLOS does not require the name of the retained segment as a parameter.

F. MAKING SEGMENTS VISIBLE (JVISBL/JDVISB)

1. Description

JVISBL/JDVISB are the subroutines that determine the visibility/default visibility of retained segments.

2. <u>Use</u>

CALL JVISBL(NAME,VISFLG)   (Immediate)

or

CALL JDVISB(VISFLG)   (Default)

3. <u>Parameter Definitions</u>

NAME   (Integer)

The name of the retained segment whose visibility attribute will be changed.

VISFLG   (Integer)

An integer value that controls the visibility of retained segments. (VISFLG=0 or 1)

Default value is one (visible).

4. <u>Discussion</u>

The following VISFLG values apply:

0 -- Retained segment invisible

1 -- Retained segment visible

JVISBL is used to remove or restore the image of a retained segment onto the screen.

JDVISB sets the default visibility of all retained segments.

Note -- <u>Neither</u> subroutines can be called within a segment (JVISBL is <u>not</u> a current subroutine call).

Both JVISBL and JDVISB can make segments visible and invisible.

G. CLEARING THE SCREEN (JFRAME)

1. <u>Description</u>

JFRAME is the subroutine used to clear the monitor screen in preparation for a new drawing area within a program.

2. Use

CALL JFRAME (No parameters)

3. Discussion

JFRAME will cause all temporary segments to be removed from the screen. All retained segments will be removed, then will be redrawn based upon their visibility attribute as specified by JVISBL/JDVISB.

JCLEAR will not only remove all retained segments from the screen, but will erase them from memory as well. After using JCLEAR a retained segment cannot be displayed again. Reference 1 contains a detailed description of JCLEAR.

H. PAUSING A PROGRAM (JPAUSE)

1. Description

JPAUSE is the subroutine used to pause a graphics program during execution.

2. Use

CALL JPAUSE(1)

3. Parameter Definition

1 (Integer number one)

Required by operating system device assignment.

4. Discussion

The JPAUSE statement can be inserted anywhere in a DI-3000 program, after JDINIT and before JEND. It is used to hold the image on the screen for extended viewing.

A request message will appear on the VT-100/2 screen during the pause. Depressing the return key on the terminal will cause the program to continue.

Every application program should include at least one JPAUSE statement after all segments have been created. If the program is not "paused" during execution, it will immediately exit at run-completion and the screen will clear.

VII. CONCLUSION

A. ADDITIONAL CAPABILITIES

This tutorial has only described the basic graphics features available in the War Lab. A brief discussion of additional capabilities follows. Detailed explanations can be found in Reference 1.

Graphics input can be read by using the logical input functions and the input tablet hardware. The functions request input from the operator and pass the values to the calling program. The Chart and Sketch Program, written by CAPT Tschudy and available to users in the War Lab, makes extensive use of DI-3000 input subroutines for interactive query and selective display.

Scaling can be done that results in distorted images or varied image sizes. All previous examples have been two-dimensional (2D) pictures, but the ability to create three-dimensional (3D) objects exists. The 2D or 3D images can be rotated throughout all possible configurations, and can be viewed from any translation point in space by using different modeling transformations.

Each error generated in DI-3000 is assigned a "severity level," and a "threshold" error level can be set that will either terminate or allow a program to continue. This selective error processing feature is particularly helpful during program development and debugging.

System inquiries can be made that pass current values, default values, modes, and status information back to the application program. This capability is useful in interactive programs that rely on dynamic parameters.

Calling programs can "escape" to the Ramtek hardware itself for control of device-dependent routines. These escape functions often complete processes more quickly by using hardware routines rather than DI-3000 software. The immediate display of all screen "pixel" data, rather than a slow software update, is an example of a typical escape function.

B. HELP FEATURE

The operating system provides users with a DI-3000 "help" feature. Printouts that discuss common graphics problems, and solutions to typical difficulties, are displayed on the terminal screens. To access the "help" library, after log-in type:

```
HELP DI3000
```

A menu will appear with topic subsections that can then be specified by typing the given topic name.

In general, "help" listings are designed to aid more experienced users.

C. SAMPLE PROGRAMS

Sample programs exist that can be used to compare displayed images to the Fortran listings that create them. Users can output the listings to their terminal screens or to the printer in the War Lab machine room. The graphics images will appear on a monitor pair that is selected in response to an interactive query during program run.

To run a demonstration program, after normal user log-in type the following command:

```
RUNDEMO (Sample Program Number)
```

For example, if sample program number 2 is to be run and displayed on a monitor, the following complete command is typed and entered with a <CR> (carriage return):

```
RUNDEMO 2
```

A request for monitor number will then appear on the terminal screen. After typing the desired monitor number and entering it, the demonstration program will run and display the image. The Ramtek monitor you choose must be energized and cannot already be in use.

There are several ways to exit, or end the programs. Most require a <CR> response to a pause message, but demonstration programs 21-25 contain FORTRAN PAUSE commands that require the following entry:

```
C <CR>
```

All programs can be terminated at any time by typing <CTRL> and Y simultaneously.

To cause the Fortran program listing to appear on the terminal screen, type the following:

```
TYPEDEMO (Sample Program Number)
```

To stop the screen from "scrolling" past the program portion you wish to view, depress the <NO SCROLL> key.

To print a hard copy of the Fortran listing, type the following:

```
PRINTDEMO (Sample Program Number)
```

Some of the sample programs correspond to figures in this tutorial, and are indicated as such in the following sample program index:

DEMO NUMBER	DESCRIPTION
1	Absolute square
2	4 Relative squares
3	Successful polygon overlay
4	Absolute broken line (Figure 2.1)
5	Relative broken line (Figure 2.1)
6	Arc line
7	Distorted arc (center off screen)
8	Polygon (Figure 2.2)
9	Rectangle
10	Filled circle
11	Circles (Figure 2.3)

12	Circle section (Figure 2.5)
13	Text, default attributes
14	Text, large letters
15	Text and polygon, normal size
16	Text, different font
17	Text, vertical path
18	Text, wide gap
19	Text, overlap gap
20	Text, base and plane change
21	Text, italics
22	Sine waves plot
23	Text, base line
24	Text, transformations
25	Interactive Input

1. Suggested Thesis Topics

A student with a desire to work with the graphics system could contribute to War Lab system capability. Polygon overlay and 3D transformation features could be more thoroughly investigated.

An extensive color table needs to be created. Some type of interactive method for users to select desired colors could be developed.

This tutorial could be made into an interactive program, and further documentation of the advertised DI-3000 capabilities that are not applicable to the War Lab system could be generated.

LIST OF REFERENCES

1. DI-3000 User's Guide, Precision Visuals, Inc., Boulder, CO, September 1982.

2. VAX-11 FORTRAN Language Reference Manual, Digital Equipment Corp., Maynard, MA, April 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center	2
	Cameron Station	
	Alexandria, Virginia 22314	
2.	Superintendent	2
	Naval Postgraduate School	
	ATTN: Code 0142	
	Monterey, California 93943	
3.	Prof. M. K. Sovereign, Code 74	1
	Chairman, C3 Academic Group	
	Naval Postgraduate School	
	Monterey, California 93943	
4.	CDR G. R. Porter, Code 741	10
	Director, C2 War Lab	
	Naval Postgraduate School	
	Monterey, California 93943	
5.	Dr. A. M. Zied, Code 74Zi	5
	Technical Director, C2 War Lab	
	Naval Postgraduate School	
	Monterey, California 93943	
6.	Joint C3 Curricular Office, Code 39	1
	Naval Postgraduate School	
	Monterey, California 93943	
7.	MAJ H. W. Yellen, USA	2
	Center for War Gaming	
	U .S. Army War College	
	Carlisle Barracks, Pennsylvania 17013	
8.	Naval Ocean Systems Center	1
	ATTN: Code 8302	
	271 Catalina Blvd.	
	San Diego, California 95152	
9.	LT Ronald H. Elmlinger, USN	3
	230 Betz Road	
	Columbus, Ohio 43207	

ATE
LME